LiDAR Based Perception System: Pioneer Technology for Safety Driving

LiDAR Based Perception System: Pioneer Technology for Safety Driving

By

Zhongzhen Luo, B.Eng.

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

McMaster University

Doctor of Philosophy (2017)

Hamilton, Ontario (Department of Computing and Software)

TITLE: LiDAR Based Perception System: Pioneer Technology for Safety Driving

AUTHOR: Zhongzhen Luo (McMaster University)

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

NUMBER OF PAGES: xx, 213

Abstract

Perceiving the surrounding multiple vehicles robustly and effectively is a very important step in building Advanced Driving Assistant System (ADAS) or autonomous vehicles. This thesis presents the design of the Light Detection and Ranging (LiDAR) perception system which consists of several sub-tasks: ground detection, object detection, object classification, and object tracking. It is believed that accomplishing these sub-tasks will provide a guideline to a vast range of potential autonomous vehicles applications. More specifically, a probability occupancy grid map based approach was developed for ground detection to address the issues of over-segmentation, under-segmentation and slow-segmentation by non-flat surface. Given the non-ground points, point cloud clustering algorithm is developed for object detection by using a Radially Bounded Nearest Neighbor (RBNN) method on the static Kd-tree. To identify the object, a supervised learning approach based on our LiDAR sensor for vehicle type classification is proposed. The proposed classification algorithm is used to classify the object into four different types: “Sedan”, “SUV”, “Van”, and “Truck”. To handle disturbances and motion uncertainties, a generalized form of Smooth Variable Structure Filter (SVSF) integrated with a combination of Hungarian algorithm (HA) and Probability Data Association Filter (PDAF), referred to as GSVSF-HA/PDAF, is developed. The developed approach is to overcome the multiple targets data association in the content of dynamics environment where the distribution of data is unpredictable. Last but not the least, a comprehensive experimental evaluation for each sub-task is presented to validate the robustness and effectiveness of our developed perception system.

Acknowledgements

First and foremost I would like to express my sincere gratitude to my advisors Dr.Martin von Mohrenschildt and Dr.Saeid Habibi for providing me with the opportunity to pursue PhD degree. Dr.Martin von Mohrenschildt has taught me both consciously and unconsciously on how good programming skills are. It is not often to find an advisor that always finds the time for listening to the little problems and roadblocks that unavoidably crop up in the course of performing research. His technical and editorial advice was essential to the completion of this thesis and has taught me innumerable lessons on my research. Also I am deeply indebted to my co-supervisor Dr.Saeid Habibi for his fundamental role in my doctoral work. He provided me with every bit of guidance, assistance, and expertise that I needed during my PhD study. Without his valuable inspiration, advice, and encouragement, this research would not have been achievable. I appreciate all his contributions of time, ideas, and lessons to make my PhD experience more productive.

Besides my supervisors, I would like to thank my committee members: Dr.Ridha Khedri and Dr.Fengjun Yan, for their insightful comments and encouragement, but also for the hard questions and vast knowledge in several discussions which guided me through this important period of my life.

My sincere thanks also goes to our industry expert: Cam Fisher. Without his precious support and thoughtful guidance, it would not be possible to access research facilities and conduct this research.

I am also grateful for my colleagues from CMHT, for the endless discussions and collaborations, and for all the days we were working together, and for all the fun we have had in the last five years.

Last but not the least, I would like to thank my parents for supporting me spiritually throughout my studies aboard and my life in general.

Contents

List of Figures

List of Tables

List of Abbreviations

LiDAR	**Li**ght **D**etection and **R**anging
NHTSA	**N**ational **H**ighway **T**raffic **A**dministration
BLS	**B**lind **S**pot **I**nformation **S**ystem
TLD	**T**racking **L**eaning **D**etection
MTT	**M**ultiple **T**arget **T**racking
ACC	**A**daptive **C**ruise **C**ontrol
FOV	**F**ield **O**f **V**iew
VDC	**V**olts **D**irect **Current**
ADAS	**A**dvanced **D**river **A**ssistance **S**ystem
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
PDDP	**P**rincipal **D**irection **D**ivisive **P**artitioning
MST	**M**inimum **S**panning **T**ree
RANSAC	**RAN**dom **S**ample **C**onsensus
GPINSAC	**G**aussian **P**rocess **IN**cremental **SA**mple **C**onsensus
NN	**N**earest **N**eighbor
DBSCAN	**D**ensity **B**ased **S**aptial **C**lustering of **A**pplications with **N**oise
RBNN	**R**adially **B**ounded **N**earest **N**eighbor
DBI	**D**avies **B**ouldin **I**ndex
DI	**D**unn **I**ndex

PCA	**P**rinciple **C**omponent **A**nalysis
SVM	**S**upport **V**ector **M**achine
ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
FCN	**F**ully **C**onvolution **N**etwork
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
ReLU	**Re**ctified **L**inear **U**nits
C-SVC	C-**S**upport **V**ector **C**lassification
RBF	**R**adial **B**asis **F**unction
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
TN	**T**rue **N**egative
FN	**F**alse **N**egative
NPV	**N**egative **P**redictive **V**alue
FPR	**F**alse **P**ositive **R**ate
FDR	**F**alse **D**iscovery **R**ate
FNR	**F**alse **N**egative **R**ate
KF	**K**alman **F**ilter
EKF	**E**xtended **K**alman **F**ilter
SVSF	**S**mooth **V**ariable **S**tructure **F**ilter
GSVSF	**G**eneralized **S**mooth **V**ariable **S**tructure **F**ilter
GVBL	**G**eneralized **V**ariable **B**oundary **L**ayer
HA	**H**ungarian **A**lgorithm
PDAF	**P**robability **D**ata **A**ssociation **F**ilter
JPDAF	**J**oint **P**robability **D**ata **A**ssociation **F**ilter

IMM	**I**nteracting **M**ultiple **M**odel
RMSE	**R**ooty **M**ean **S**quare **D**eviation
CT	**C**orrectly **T**racked
FT	**F**alsely **T**racked
GUI	**G**raphical **U**ser **I**nterface
UDP	**U**ser **D**atagram **P**rotocol
IDE	**I**ntegrated **D**evelopment **E**nvironment

Declaration of Authorship

I, Zhongzhen LUO, declare that this thesis titled, “LiDAR Based Perception System: Pioneer Technology for Safety Driving” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

“*The first autonomous cars date back to the late 20th century. But recent increases in sophistication and reductions in cost - reflected, for example, in cheap LIDAR systems, which can 'see' a street in 3D in a way similar to that of the human eye - are now bringing autonomous cars closer to the market.*”

Carlo Ratti

Chapter 1

Introduction

1.1 Motivation and Objectives

Autonomous vehicles are a promising evolution of current vehicle technology for enhanced road safety, improved road efficiency, reduced cost of congestion and decreased fuel consumption, while improving mobility and liability and hence accident saving. The self-driving demo car has been getting much attention, due to significant development efforts and dramatic progress made by companies such as Google. Research on autonomous vehicles has been growing rapidly in recent years and encompasses different domains, including robotics, computer science, and engineering. Furthermore, it should be noted that scientific advances have been made by car manufacturers who do not always publicly disclose the details on their approaches or algorithms, owing to commercial sensitivity. Although general use of autonomous vehicles for widespread use on public roads is likely years away, these vehicles are already being operated in a limited form on highways or suburban district.

Autonomous vehicles are partially or fully controlled by computer programs, which may eventually require no human driver at all. Technological advancements have made possible the progression from traditional human-driven vehicles to completely autonomous vehicles. National Highway Traffic Administration (NHTSA) has established a definition of autonomous vehicles by level of automation [1]. According to NHTSA, the definitions are summarized as the following:

- Level 0 (No Automation) - The driver is in complete control of the primary vehicle controls at all times such as brake, steering, throttle, and motive power.

- Level 1 (Function-specific Automation) - One or more specific control functions are automated independently. The driver is fully engaged and responsible for overall vehicle control.

- Level 2 (Combined Function Automation) - At least two control functions are automated such as adaptive cruise control in combination with lane keeping. The driver disengages from active control in certain limited driving situations, and is still responsible for monitoring the roadway and safe operation.

- Level 3 (Limited Self-Driving Automation) - The driver cedes full control of all safety-critical functions under certain traffic or environmental conditions, relying heavily on the vehicle to sense changes in those conditions that require the driver to take back control within a comfortable transition time.

- Level 4 (Full Self-Driving Automation) - The vehicle is designed to perform all safety-critical driving functions and monitor roadway conditions for an entire trip. The driver is not expected to operate at any time or else the vehicle can be unoccupied.

Following the regulation guidelines by NHTSA, current vehicles are increasingly moving toward the goal of simplifying the driver's job and automating parts of the driving process. For instance, the newest 2017 Cadillac ATS introduces a driver assist package that includes automatic collision preparation, front and rear automatic braking, full-speed range adaptive cruise control, lane keep assist, front pedestrian detection and rear cross-traffic alert [2]. Based on the level of automation by NHTSA, the current level of automation for Cadillac is Level 2, slowly approaching Level 3.

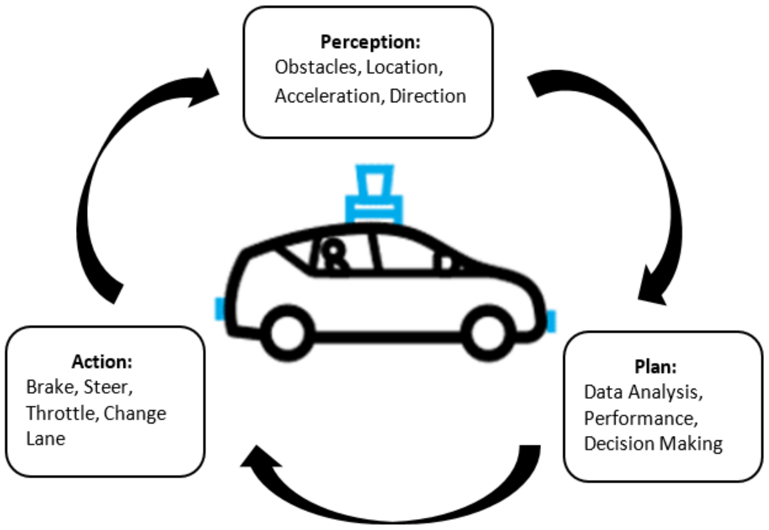

Perception:
Obstacles, Location, Acceleration, Direction
Plan:
Data Analysis, Performance, Decision Making
Action:
Brake, Steer, Throttle, Change Lane

FIGURE 1.1: The cycle of autonomous vehicle system

Among the many technologies which make autonomous vehicles possible is a combination of sensors and actuators and powerful processors. The autonomous vehicle employs "Perception-Plan-Action" cycle which is the fundamental to many other intelligent system (See Figure 1.1). For a fully autonomous vehicle system, the sensors firstly percept the environment and location of the vehicle. Then the developed algorithms interpret and process the sensory information, identify the situations and plan the trajectories so as to exert the full control such as brake, steer, change lanes or throttle. There are dozens of subsystems and hundreds of specialized sensor channels for these three steps. The four different types of sensors will help to provide external and immediate information to autonomous and semi-autonomous vehicles:

- Radar - the object detection system that uses radio waves to determine the range, angle, or velocity of objects.
- Ultrasonic - the object detection system which emits ultrasonic sound waves and detects their return to determine distance.
- Camera - the use of passive cameras and sophisticated object detection algorithms to understand what is visible from the cameras.
- LiDAR - a surveying technology that measures distance by illuminating a target with a laser light.

Many vehicles now have employed radar sensors for obstacles detection due to small size, inexpensive and good range (See Figure 1.2A). They also work equally well in dark conditions, and are able to better sense through fog, rain, and snow. Radar is very effective at determining relative speed of objects. Although the size of radar makes it better for near-proximity detection, poorer resolution than other sensors limits the usage to stationary objects and precise range measurement. Ultrasonic sensors actively emit high frequency sound (See Figure 1.2B).

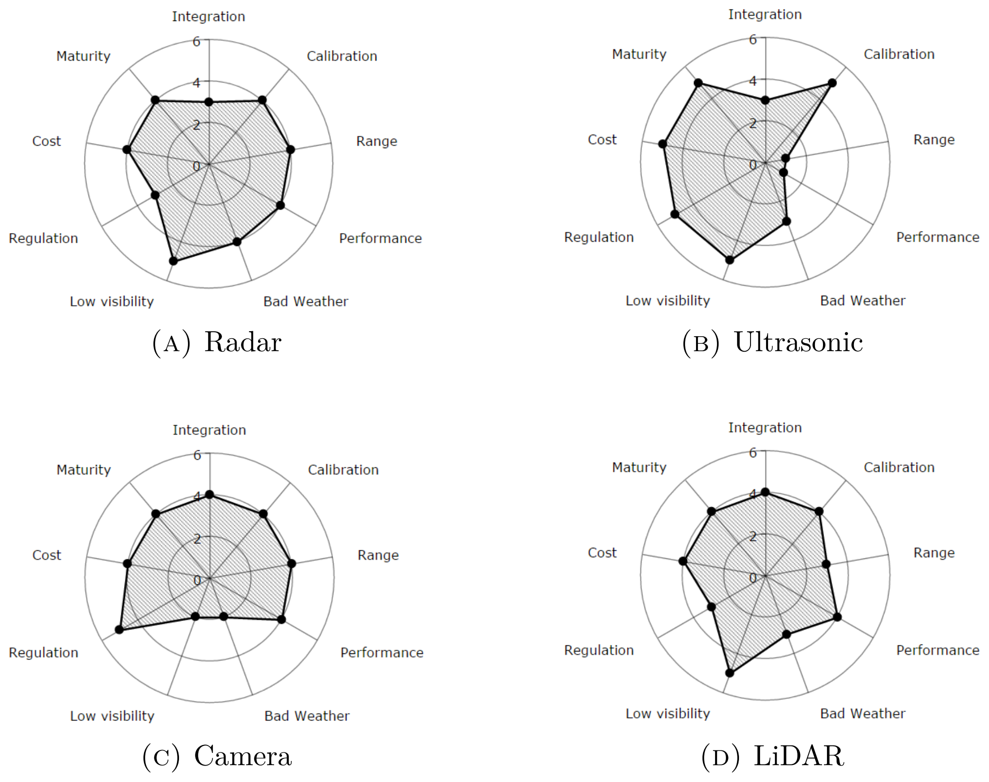
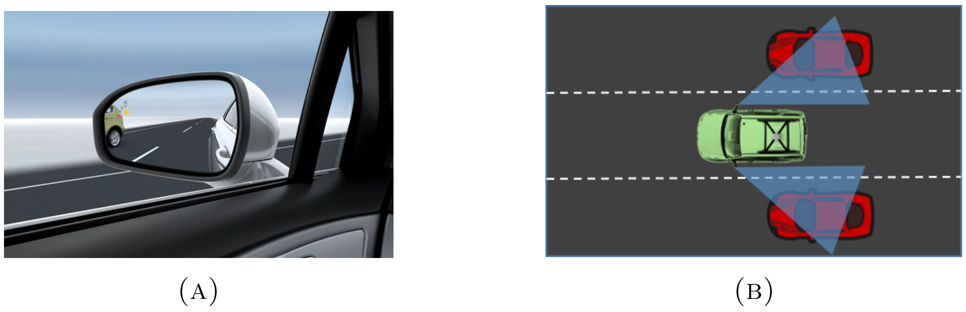


FIGURE 1.2: Sensors comparison in autonomous driving

As sound waves are comparatively slow, so measurement in less than centimeter are detectable. So these sensors are excellent for very-near-range three-dimensional mapping, and they are effective regardless of light conditions. Due to their short range, they also work well in conditions of snow, fog, and rain. However, they are not useful for measurement of speed because of their short range. Vision based sensors have become very cheap, small, and with increasing resolution in recent years (See Figure 1.2C). Their color, contrast, and optical-character recognition capabilities give them a new powerful feature compared to other sensors. But they are less useful for very close proximity detection. For longer distance, their range and performance degrades, as human eyes do, contingent on light intensity. In very bright conditions, it is possible for some implementations to not identify objects against bright skies, which was reported as the main cause in the May 2016 Tesla Autopilot-related fatality in Florida.

Light Distancing and Ranging (LiDAR), on the other hand, are currently large and expensive systems but increasingly used in the autonomous vehicle applications (See Figure 1.2D). For example, the Google self-driving car has mounted and utilized a LiDAR senor on its roof. Current implementations have improved range substantially from 20 meter up to 200 meter which has turned cars into machines capable of mapping their surroundings with high resolution as well. Current LiDAR sensors work well in all light conditions, but start failing in snow, fog, rain, and dust particles in the air. At present, despite the fact that LiDAR is a superior technology, there is universal agreement that the technology is currently too expensive for wide deployment. However, the difference in price between LiDAR and other sensors could become negligible as the technology evolves. Quanergy system promises to release a solid-state LiDAR in the next couple of years that will be less than $100, bringing costs in line with cameras or radars while offering better perceptive ability.

For autonomous driving, LiDAR will never be a standalone solution. The system will most likely be used in conjunction with other sensors such as cameras to identify color. Because the LiDAR sensor is superior at detecting and measuring object, utilizing its own strengths will be more reliable which making them a requirement for a complete autonomous system.



(A) (B)

FIGURE 1.3: Ford's blind spot information system

1.2 Problem Description

One of the arguments for autonomous vehicle is that they will be safer than conventional vehicles. According to NHTSA [3], 94% of car crashes are caused by human error which include speeding, distraction, drunk driving and so on. In recent years, although amount of active safety technologies (eg., advance driving assistance system, adaptive cruise control) have been developed for intelligent vehicle or self-driving, the number of traffic accidents because of individual differences in the way of driving are still significant. For example, the new 2017 Ford Fusion Series has been equipped with radar sensors for Blind Spot Information System (BLIS) which is designed to detect vehicles that may have entered the blind spot zone (See Figure 1.3a). The detection zone is on both sides of the vehicle, extending rearward from the exterior mirrors to about 3 meters beyond the bumper (See Figure 1.3b). The system aims to alert the driver if certain moving vehicles enter the blind spot zone while driving. However, the BLIS has limitations that may limit BLIS detection including [4]:

- Certain maneuvers of other vehicles as they enter and exit the blind zone.

- Vehicles passing through the blind zone at very fast speeds.
- Several vehicles forming a convoy and passing through the blind zone.

Over the past few years, Google autonomous vehicles have been involved in a number of minor accidents, but up until now all of them were caused by other human driven vehicles not Google's vehicle itself [5]. When Figure 1.4 shows some highway scenarios that the car accident is likely to occur even when equipped with cutting-edge detection systems. In Figure 1.4A, for example, both the red car and the green car are planing to change lane to the middle. In this case, the passive detection system would not trigger an alert since there is no vehicle in the detection zone until the distance between two cars becomes small. If this happens in a highway situation, it is too late for the driver or the autonomous vehicle to react because of their high speed. Likewise, two cars may collide as the failure of detection system in Figure 1.4B. Figure 1.4C illustrate certain maneuvers such as zig-zag driving which will also put nearby vehicles into dangerous spot.

Therefore a comprehensive environment perception in a safe manner that detects, classifies and estimates the motion states of surrounding vehicles would have an enormous potential to enhance road safety by proactively reacting to traffic conditions or hazardous situations. Bear in mind that there exists no sensor which can perceive the intentions of drivers. Since direct observation of environmental states are limited to position, velocity and other basic geometric features and therefore processing of acquiring data is needed. Hence reliable and efficient perception of the dynamic and complex environment is one of the unsolved problems in the autonomous vehicle application. Specifically, the ability to identify the surrounding

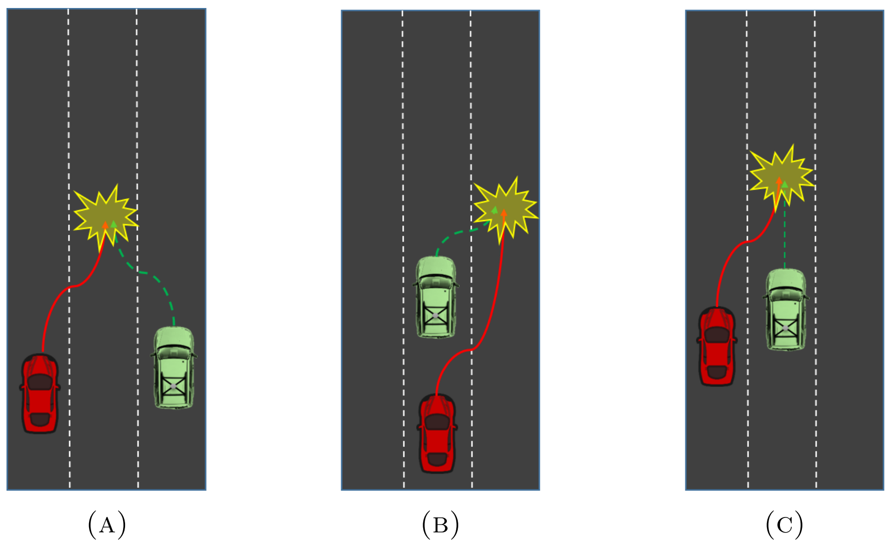


Figure 1.4: Some highway scenarios of intelligent or autonomous vehicle crashes

vehicles, pedestrians, road lanes, traffic lights, traffic signs, and others are critical to performance [6].

To realize the potential of autonomous driving and to achieve proactive accident avoidance, the system needs to be as reliable and effectively as possible. These requirements are particularly important due to complexity, diversity of environments and varying choices for driver response in urban traffic. One of the limiting factors is usually the time budget for decisions. If the intervention strategy of a vehicle system clashes with the intentions and actions of the driver, the resulting ambiguities could lead to delays, missing the window of opportunity for mitigating a traffic conflict. Hence, inferring the driver's intentions and predicting their responses to a hazardous situation as early as possible are of central importance for coordinated driver assistance. As such, the expected outcomes can be explained by the following table which specify what the system must do. As requirement specification can be view as a high-level design specification of the system to be built, so the expected results from our system are listed in Table 1.1 according to the following priority levels.

Priority Definitions: The following definitions and Table 1.1 are intended as a guideline to prioritizing requirements.

- Priority 1 – The requirement is a “must have” as outlined.
- Priority 2 – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits.
- Priority 3 – The requirement is a “nice to have” which may include new functionality.

Reference	Functional Requirement	Priority
RF01	The system shall provide three dimensional point cloud map of surrounded environment on screen.	1
RF02	The system shall detect the ground points in a non-planar surface	1
RF03	The system shall detect object without priori knowledge within field of view.	1
RF04	The system shall category object into different types.	1
RF05	The system shall track motion states of nearby vehicles simultaneously and provide tracked vehicles' information such as relative speed(h/km), relative distance(m).	1
RF06	The system shall yields an estimated driving direction of multiple vehicles to alert the driver for proactive avoidance.	1
RF07	The system shall be capable of operating on both online and offline mode.	2
RF08	The system shall be capable of storing real time data to hard drive for offline playback.	2
RF09	The communication time and data processing time of the system shall be less than the measurement device sampling rate (10Hz).	1

Table 1.1: Functional requirement of the system

1.3 Novelty of the Research

In this research, given the measurements coming in the form of a point cloud from a LIDAR mounted on the roof of a vehicle, a perception system is developed in order to obtain a consistent and meaningful representation of the environment. The design of our perception system is decomposed in different sub-tasks.

1. Ground Detection
2. Object Detection
3. Object Classification
4. Object Tracking
5. Real Time System Implementation

Figure 1.5 demonstrate the process of our perception system that to solve each sub-tasks in sequence. Although parts of the sub-tasks have been done before, it does not appear that anyone has ever brought it all together into one single project. In this following chapters, the design of algorithms are discussed to solve the sub-tasks step by step.

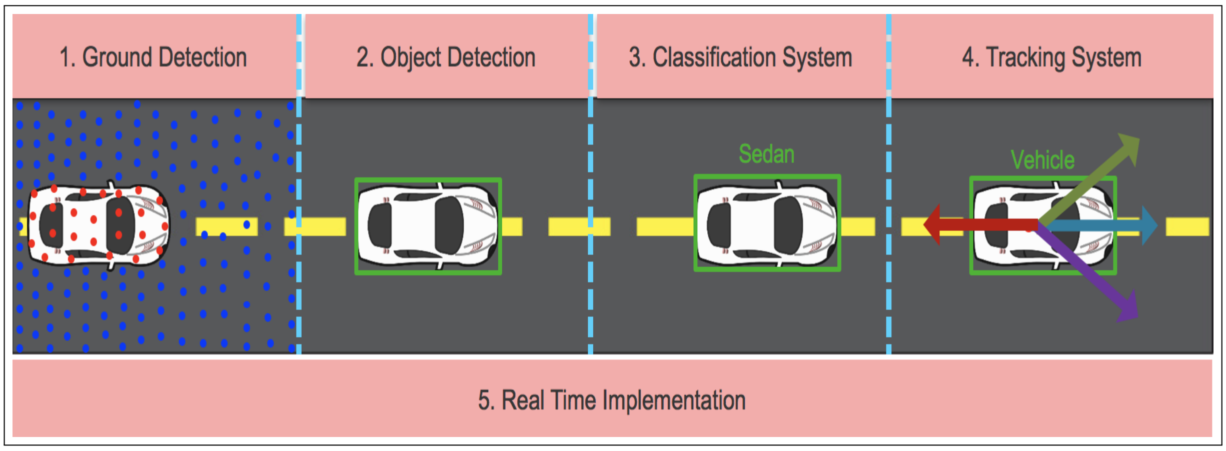


FIGURE 1.5: The process of our perception system

The contributions of this thesis will cover multiple areas.

The first contribution of this research is the design of a novel framework that decomposes the long-term task into several sub-tasks: segmentation, clustering, classification, tracking and estimation. Each sub-task is addressed by using several novel algorithms.

The second contribution of the search is the innovative strategy used for identifying both flat ground and non-flat ground. This fast and effective techniques provides a significant improvement in terms of the accuracy for overall system performance.

The third contribution follows the ground segmentation in which the non-ground points are obtained. We present clustering and classification methods to detect, recognize and identify multiple moving objects. The captured discriminative information of each identified vehicle will help to improve the estimation of motion states.

The LiDAR data includes both dense data and sparse data, which is challenges to perception system, i.e., deciding which measurement is associated to which target and which measurements are clutter originated. To address this problem, the forth contribution of the research is a proposed generalized smooth variable structure filter using a hybrid approach for multiple targets tracking and estimation.

Last but not the least, the fifth contribution of the research is the implementation of the real time system by illustrating how to build a real-time long-term system based on this framework. It is also shown that the developed system could be extended for multiple sensors' fusion.

1.4 Overview of the Thesis

The following is a brief account of the contents of this thesis. Chapter 2 provides an introduction to sensor technology and implementation the LiDAR and our real time plat form. The application of LiDAR sensor in autonomous industry is discussed. Chapter 3 introduces the design of ground detection in detail. The Chapter 4 follows the result of Chapter 3 and the design of object detection is discovered. Chapter 5 introduces the design of the classification system. Several machine learning techniques are described and used in this chapter. Chapter 6 will demonstrate the design of our tracking and estimation system. Two different data association techniques that are used for ether dense and sparse LiDAR data are discussed. Chapter 7 details the design and implementation of our proposed perception system. Both MATLAB and C++ versions of the algorithms are presented to illustrate the development and prototyping of our system and its real time application. Chapter 8 provides the conclusions and contributions of the thesis.

Chapter 2

Technology: LiDAR Sensor

2.1 Introduction

The sensors provide sensed information for representing the surrounding environment. Therefore, they are of the most important factors for decision making for autonomous vehicles. This chapter considers the latest research on vehicle perception systems. It also introduces the setup of our LiDAR sensor and our experimental platform.

Sensors based perception systems require sophisticated algorithms. For instance, Zdenek developed famous Tracking-Leaning-Detection (TLD) algorithm for real time tracking of unknown objects in video streams [7]. The TLD works very well when the motions of object is inside a bounding box are consistent. However, if the transformations are complicated or the bounding box includes a large portion of the background, TLD is likely to fail in the object tracking the task. Although camera based systems have outstanding features with low cost, they are less effective under complex circumstance such as bad weather conditions [8]. They must also deal with light intensity variations and they need illumination at night where headlights might not be enough.

Gokhan *et al.* [9] presented performance comparison of several tracking algorithms for a ground based radar. Most of the existing perception techniques using Radars have been compared in terms of track loss and estimation errors. Among these published strategies [10, 11, 12], Radar detection and tracking using the Interacting Multiple Model Probabilistic Data Association Filter (IMM-PDAF) algorithm with no a priori knowledge of the target motion has performed better than all other algorithms even compared to the ones that assume knowledge of the target maneuverer. Lingmin *et al.* [13] combined the conventional Kalman filtering techniques with probabilistic data association methods to improve the detection by radar. However, his algorithm could only apply to one track. In [14], Glenn *et al.* discussed the technical and practical issues in the integration of a fully operational Adaptive Cruise Control (ACC) system on a vehicular platform by using either the Radar or the LiDAR technology, as operating under the exact same environmental conditions (e.g.: traffic patterns, roadway geometry/surface, weather conditions, etc.). The main drawback of radar is narrow field of view and reduced angular resolution compared with LiDAR [15].

The LiDAR has been widely used in autonomous applications due to its high precision and range information to detect object with wide fields of view, low processing requirements and its capability in adverse weather conditions [16, 25]. In [17], Dominguez *et al.* present a LiDAR based perception solution for autonomous vehicles. The LiDAR sensor captures and represents objects by measuring position and elevation of each single point as three dimensional coordinates. The points are recorded in an unorganized structure due to the nature of the scan, requiring specific data organization and processing techniques which are significantly different from the existing image processing algorithms. Jaebum Choi *et al.* [18] present a robust algorithm for Multiple Target Tracking (MTT) using a 3D LiDAR. They introduced feature based object geometry for precise estimation of the system state in order to compensate the unintended dynamics caused by shape change or occlusion. A real time intersection detection approach is proposed in [19] based on 3D LIDAR. Two different shaped of intersections are distinguished with the performance of above 80% at a real-time classification rate of 5Hz.

2.2 LiDAR

The word "lase " stands for "light amplification by stimulated emission of radiation‘'. A laser is a device which generates a stream of high energy particles (photons) within an extremely narrow range of wavelengths. A laser light source forms the basis for a LiDAR system. The wavelength chosen for most lasers is 1064 nanometers, which is in the near-infrared band of the electromagnetic spectrum [20].

The essential measurement made by a LiDAR sensor is of time, the time that elapses from the moment the pulse is emitted until it returns after being reflected by the target surface. The laser pulse travels at the speed of light, and time can be directly converted to distance, by the following equation

$$\text{Distance} = \frac{(\text{elapsed time}) \times (\text{speed of light})}{2} \tag{2.1}$$

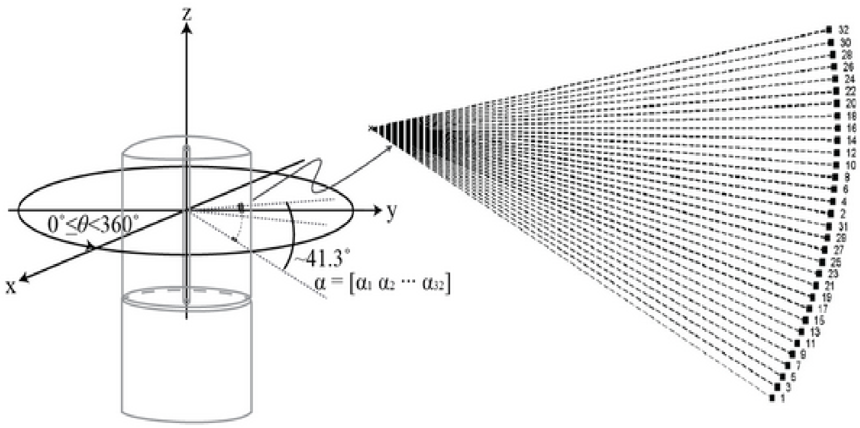


FIGURE 2.1: 32 radially oriented lasers embedded inside the LiDAR sensor

The Velodyne HDL-32E LiDAR, depicted in Figure 2.1, is an ultra compact and cost effective LiDAR sensor. The dimensions of this sensor is 8.5cm $\times$ 8.5cm $\times$ 15cm$(L \times W \times H)$ and the net weight is 1.3 kg. The HDL-32E comprises a vertical array of 32 radially-oriented lasers with an effective 41.3° vertical field of view (FOV) from +10.67° to -30.67°. This entire sensor can spin about its vertical axis at speeds of 10 Hz, resulting in a full 360° azimuthal field of view. According to the user manual [20], the limitation of range measurement is up to 70m and the sensor generates 700,000 points per second. The detail specification of HDL-32E LiDAR is shown at Table 2.2.

Specification	Velodyne HDL-32E LiDAR
Laser	• 905 nm wavelength • Time of flight distance measurement • Measurement range 70m (1m up to 70m)
Sensor	• 32 laser/detector pairs • $-30.67°$ to $+10.67°$ vertical FOV • 360° horizontal FOV • 10 Hz frame rate • Operating temperature: $-10°C$ to $+60°C$ • Storing temperature: $-40°C$ to $+105°C$ • Accuracy: < 2 cm • Vertical angular resolution: 1.33° • Horizontal angular resolution: 0.16° at 600 RPM
Mechanical	• Dimension: 149.86mm x 85.3mm • Weight: 1kg (plus 0.3kg for cabling)
Output	• Approximately 700,000 points/second • 100 Mbps Ethernet connection • UDP packets (distance, intensity, rotation angle)

Table 2.1: LiDAR Specification

Because our LiDAR produce User Datagram Protocol (UDP) network packets to transmit data over Ethernet for each of the lasers in the device. The structure of UDP packet generated by our LiDAR is shown in Appendix A1.

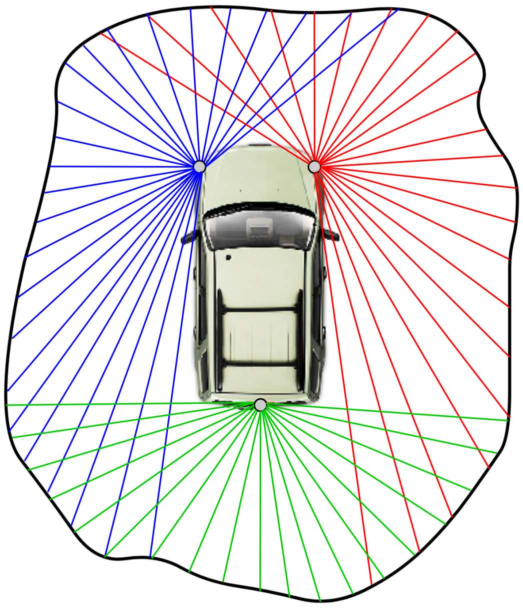
Let one frame be defined such that a LiDAR has scanned a full 360° coverage. With UDP packets decoding, a set of point cloud data from each frame can be denoted by P which is given by:

$$P = \left\{ \begin{bmatrix} \rho_\theta^j cos(\alpha^j) sin(\theta^j) \\ \rho_\theta^j cos(\alpha^j) cos(\theta^j) \\ \rho_\theta^j sin(\alpha^j) \end{bmatrix} \middle| 1 \le j \le 32; 0 \le \theta < 2\pi \right\} \tag{2.2}$$

where α^j represent the fixed vertical angle of laser beam j. ρ_θ^j represent the range measurement from laser beam j at scanning position θ.

2.3 LiDAR Position

The fully autonomous vehicles developed by companies like Google and Baidu all rely on LiDAR to perceive and map the world. The maps are critically implemented as they provide environmental context for the vehicles and let them focus their computing power on transient obstacles like cars, pedestrians, and cyclists. There are two common LiDAR configurations in autonomous vehicles: 1) installed around the vehicle at a low height based on a planar LiDAR configuration and 2) mounted on the roof of the vehicle based on one single 3D LiDAR. In [21],Figure 2.2 presented a comparison using these two types of configuration.


(A) Two LiDARs are mounted on the two front corners, and the third one covers the back side

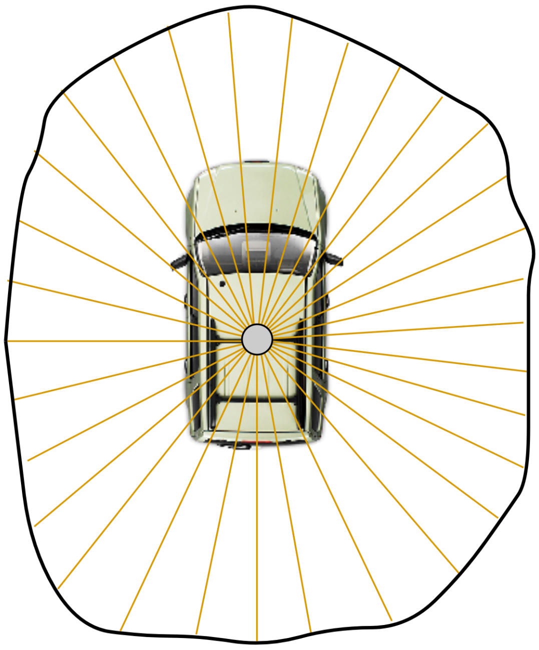

(B) One single 3D LiDAR mounted on the top

FIGURE 2.2: Two types of LiDAR configuration

From Figure 2.2, both configurations generate a 360° scan. The qualitative results from [21] are summarized in the following Table 2.1.

Attributes	Planar LiDAR	3D LiDAR
Accuracy	++	+
Robustness	+	++
Integration	+	++
Calibration	+	++
Cost	++	+
Direct 3D Perception	+	++

TABLE 2.2: Comparison between LiDARs

In the static environment, the planar LiDAR performs slightly better than the 3D LiDAR as the scans are less noisy. But an extra calibrate step is needed to merge multiple local coordinates into one global coordinate system before processing the data. There are other advantages of using a single 3D LiDAR including robustness, simplicity of installation, and general setup, as presented in [22].

The LiDAR used in our research (the HDL-32E LiDAR) is mounted vertically above the roof surface of a Ford Escape as shown in Figure 2.3.

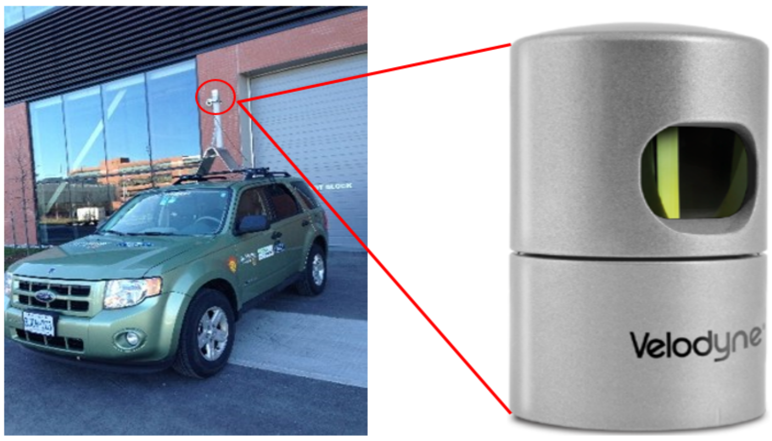


Figure 2.3: The Velodyne HDL-32E LiDAR setup

Experiments based on our configuration was performed to collect and validate the robustness and effectiveness of our developed real time system [24]. The LiDAR raw data were acquired from actual driving on the highway linking two cities: Toronto and Hamilton in Ontario, Canada. The vehicle acquired data at an average vehicle speed of about 120 km/h under a low traffic scenario, 80km/h under a medium traffic scenario and 60km/h under a high traffic scenario. The LiDAR was connected to a PC for data collection and processing which had an Intel i7 CPU and a 16 GB RAM. Due to hardware constrain, the Region of Interest (ROI) observed by our system was set to $40m \times 20m$.

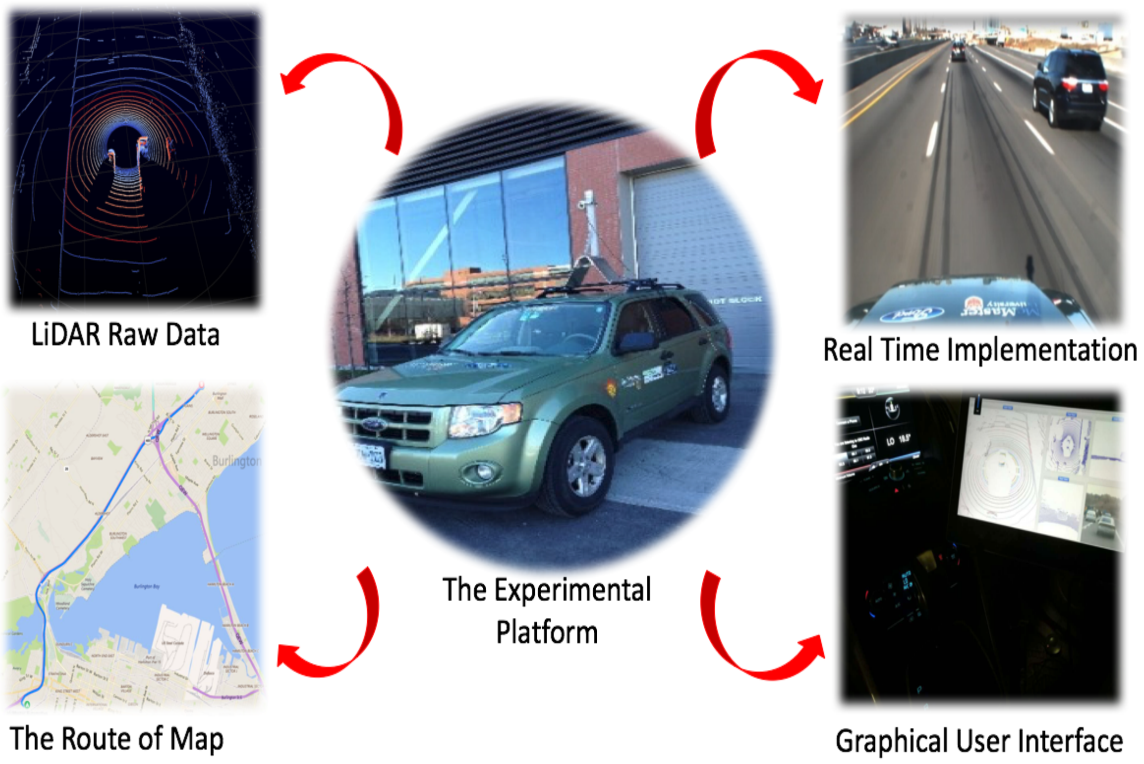


FIGURE 2.4: The real time experiment with our developed LiDAR perception system

The resulting point cloud from one single scan of our LiDAR sensor consisted of dense and sparse data as shown in Figure 2.5.

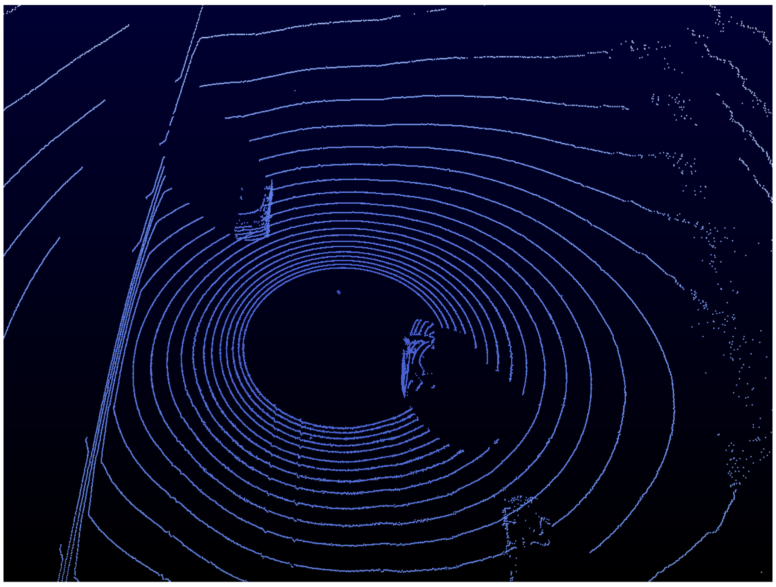


Figure 2.5: The LiDAR raw point cloud data

The developed perception system will be evaluated on different scenarios with noisy data: low traffic to high traffic as shown in Figure 2.6 and Table 2.3. Because the high traffic is characterized when a volume of traffic demand for space approaches the capacity of a road. The common scenario is vehicles driving at slower speeds, longer trip times, and increased vehicular queuing. These different driving maneuvers can change in rapid succession. Consequently, the high traffic lead to a strong degradation on the quantity and quality of data which is a great challenges for our perception system. The roadway of experiment has been conducted on the Queen Elizabeth Way expressway (QEW) linking two cities: Hamilton and Toronto and highway 6: Hamilton to Waterloo. The total acquired data size is 20 GB, the total number of LiDAR points is about 28 million. There are two computers being utilized as our processing platform as shown in Table 2.4. The first computer is used for algorithms design and implementation and the second one is used for storing data.

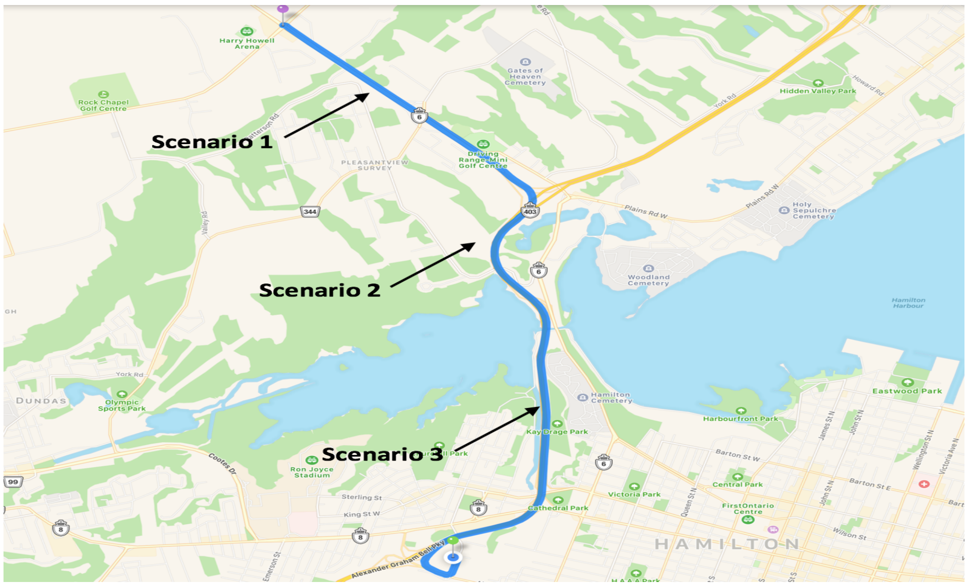


FIGURE 2.6: Three different scenarios

Scenarios	Scenario 1	Scenario 2	Scenario 3
Traffic Info	Medium Traffic	High Traffic	Low Traffic
Average Speed	60 km/h	30km/h	100km/h
Time	10 min	20 min	10min
Points	7 million	14 million	7 million
Data Size	5 GB	10 GB	5 GB
Roadway	Highway	Expressway	Expressway

TABLE 2.3: The Evaluation Experimental Scenarios

Computer	PC 1	PC 2
CPU	i7-4800 (2.7GHz)	i3-5157 (2.5GHz)
RAM	16GB	4GB
GPU	Nvidia 650M	Intel 4000

TABLE 2.4: The Processing Platform

Chapter 3

The Algorithm: Design of Ground Detection System

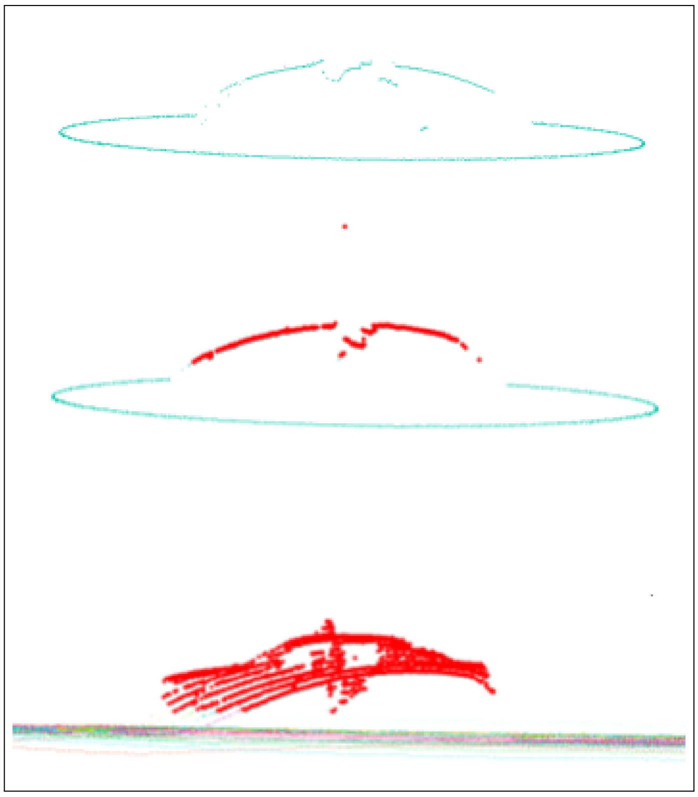


Figure 3.1: The process of ground detection system

3.1 Introduction

In this chapter, a real time gourd detection system based on a single LiDAR sensor is proposed in order to classify the ground points and model the slope of the road in various conditions. Reliable and efficient ground segmentation plays an important role in the sequence of data processing for autonomous vehicle applications, as it can help to reduce the size of data to be processed and further decrease the overall computational time. To address these characteristics, a novel method based on probability occupancy grid map is proposed to achieve high accuracy as well as efficiency. The system is composed of three sub-modules: (i) data acquisition and prepossessing, (ii) probability occupancy grid map modeling, and (iii) weighted linear regression. The core part of the whole system is the partition of data for either training or processing in every frame. The real time experiments were conducted in order to validate the effectiveness and robustness of our proposed system. Figure 3.1 show the overview of ground detection system.

The perception system for autonomous vehicles usually require a variety of sensors to acquire data such as LiDAR, radar, camera and so on. To process the vast amount of data collected from these different sensors is a great challenge for real time processing. At present, ground segmentation is a necessary intermediate step to eliminate redundant data and to reduce computation complexity. Ground segmentation is achieved by partitioning data into several sets for different functions such as tracking, prediction, or driving assistance system. Majors issue of segmentation systems are over-segmentation, under-segmentation or slow-segmentation [23]. These issues have a strong influence on the performance of the perception system [24]. For example, a false segmentation of non-ground points would result in them being classified as object (Figure 3.2). In other words, having a reliable, efficient and noise-free ground segmentation algorithm is of a great importance for the reliability and computational complexity of the system.

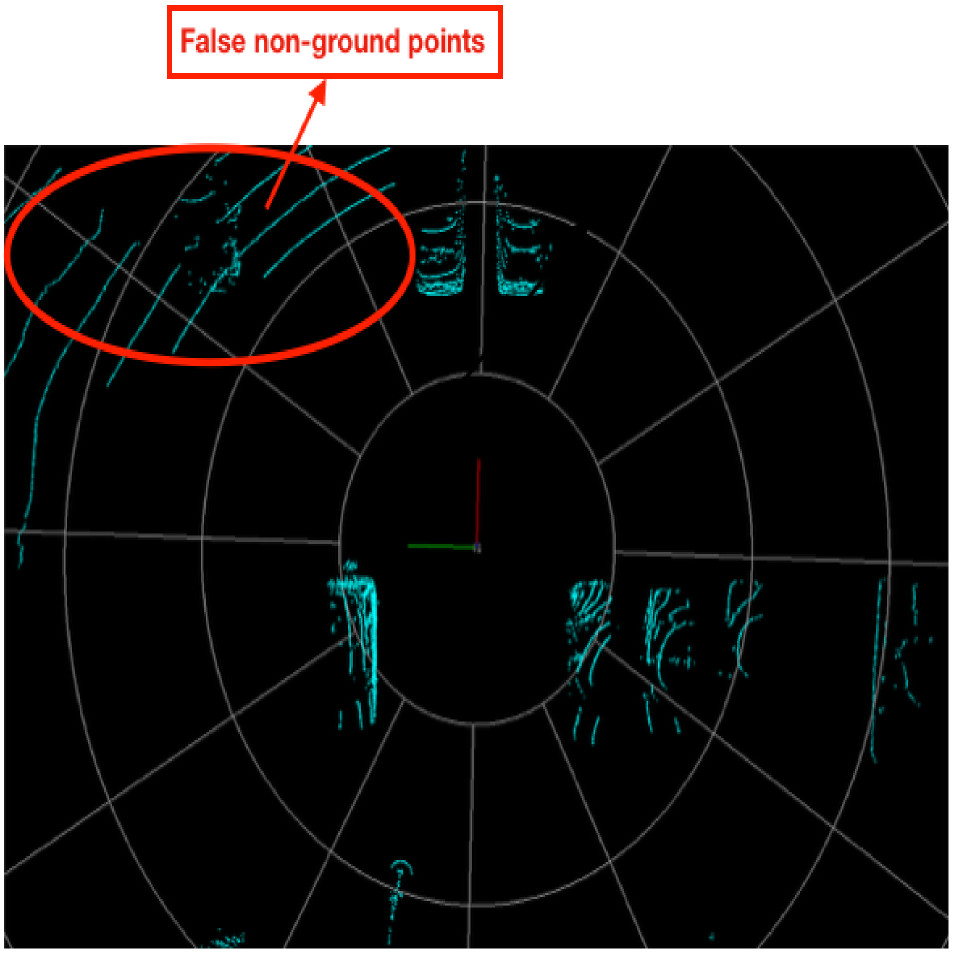


FIGURE 3.2: The example of ground points detected as non-ground points

In this chapter, a real time approach for ground segmentation is proposed and implemented to efficiently and robustly estimates the ground position and slope by taking of advantages of the geometry of the roof mounted LiDAR sensor as described in Chapter 2. The proposed real time system is based on probability occupancy grid map and weighted linear regression for ground segmentation in order to achieve high accuracy results as well as efficient computation. The proposed system is composed of three elements:

1. The raw LiDAR point cloud is partitioned into low distortion data set and high distortion data set because the points from close region has relatively lower distortion. The low distortion data is used to develop the ground estimation model. The high distortion data set is segmented based on a ground model.

2. The ground estimation model is developed by computing approximate posterior estimates from an occupancy grid map which address the problem by generating maps to represent the environment from noisy and uncertain sensor measurement data.

3. A recursively weighted linear regression to estimate the slope of the road from the low distortion data set.

4. The developed ground segmentation model is then used to segment the sparse data.

3.2 Related Work

Segmentation on grid map is one of the most popular methods that convert point cloud into a grip map and evaluate data on each grid by attributes. A simple solution is to project the data into a two and a half occupancy grid [26]. Frank and Oliver [23] presented a segmentation algorithm based on the concept of local convexity and compared their attributes such as normal vector. Kammel *et al.* [28] proposed an algorithm to assess vertical displacement in each grid. If the displacement exceeds a given threshold then the grid is marked as occupied. Himmelbach *et al.* [29] mapped data in to a polarized grid map and used a non-parametric ground model to fit ground plane. Guo *et al.* [30] manipulated the grid map as a graph by using Markov Random Field to label each grid into four different attributes: the reachable region, the drivable region, the obstacle region and the unknown region.

Point cloud segmentation on the undirected graph structure is recently another fast and efficient methods. Since each of the lasers has a fixed pitch angle, and thus would sweep out a circle of a fixed radius on a flat ground plane as the sensor rotates. Montemerlo *et al.* [27] segmented the non-ground points by comparing the range returned by two adjacent laser beams, where "adjacent" is measured in terms of the pointing angle of the beams. Sloped terrain locally compresses these rings, causing the distance between adjacent rings to be smaller on flat terrain. However, this ring-like pattern evaluation system will probably fail on the non-flat ground. Douillar *et al.* [31] proposed a set of segmentation methods designed for various densities of 3D point clouds. Among these methods, the Gaussian Process Incremental Sample Consensus (GP-INSAC) algorithm is selected with an additional outliers rejection capability for ground surface estimation. However, its computation time is unrealistic for real time application. Anguelov *et al.* [32] applied machine learning techniques to the segmentation problem under a supervised learning framework: Markov Random Field. Although the result is good, inferring point labels by Markov Random Field for large amount of data can not be done in real time.

As mentioned above, the segmentation step is the intermediate process in order to accurately classify and track potential targets. The most important contribution from segmentation is to eliminate the redundant points as much as possible in order to reduce the computation time in real time applications and improving reliability of algorithms such as artificial neural network for classification or interactive multiple models for tracking [33].

3.3 Methodology

Figure 3.3 illustrates the overall approach implemented by our real time system that is comprised of three steps: (a) data acquisition and preprocessing for extracting points from raw LiDAR data; (b) mapping and evaluation for determining the ground points and non-grounding points; and (c) weighted linear regression for estimating the slope of the road in order to guarantee fast filtering as well as maintaining higher accuracy for segmentation. These are explained in the following sections.

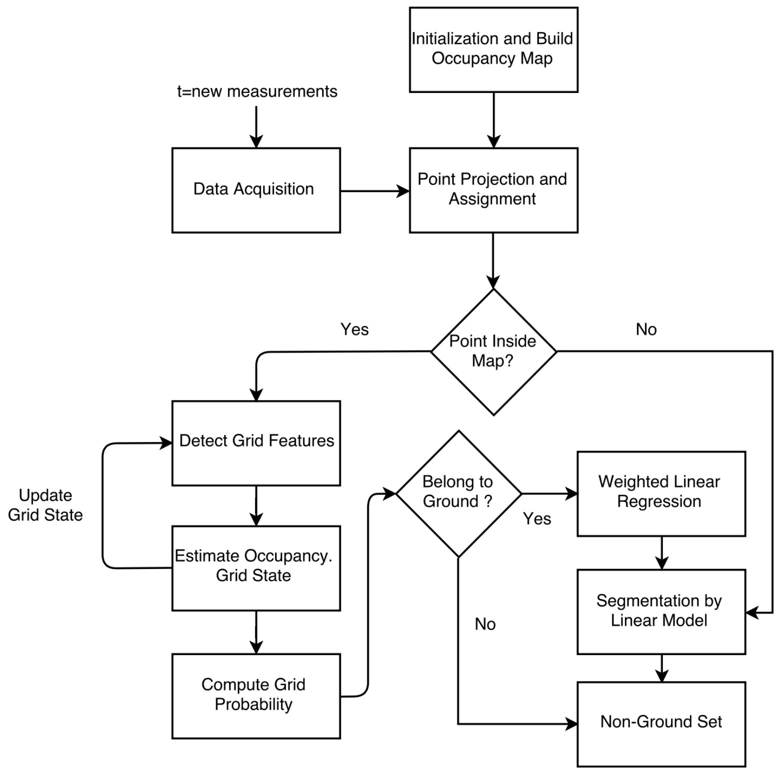


FIGURE 3.3: The data flow of ground detection system

3.3.1 Data Acquisition and Preprocessing

LiDAR data arrives in UDP packets at a rate of approximate 10 Hz via 100 MBit Ethernet. As the pitch angles of the lasers are fixed, each of the lower lasers produce a ring of point measurements in a horizontal plane. To segment the ground points, a set of raw data acquired from LiDAR via UDP packets is converted from spherical coordinates to Cartesian coordinates according to Eq.(2.2). A naive approach could discard all points which are under a certain predefined height. A slightly better approach fits a horizontal plane is accomplished through the application of the Random Sample Consensus (RANSAC) for fitting of models in the presence of many data outliers [34]. The RANSAC algorithm is very simple and it consisted of 4 steps:

1. Select random samples of to fit model.
2. Compute the model from sample set.
3. Compute the set of inliers to this model from whole data set.
4. Repeat step 1-3 until model with the most inliers over all samples is found.

The problems in both methods arises when they encounter a non-planar surface, or noisy data. Figure 3.4 shows a non trivial example in which simple fitting based filter fails.

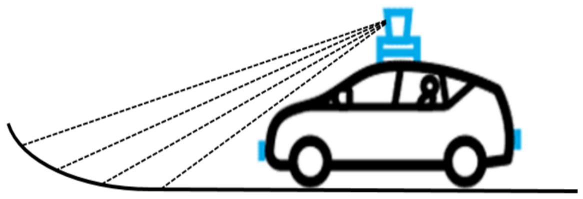


FIGURE 3.4: Non-planar surface

Given the geometry of LiDAR which is mounted on the roof of vehicle, the frame is processed through the ring-like patterns as it is the natural way for this type of sensor. Thus it is possible to estimate the slope of the ground by only processing points from inner rings given that the points from inner rings have lower distortion relative to the points from outer rings as shown in Figure 3.5.

Taking advantage of the geometry of the LiDAR, reasonable and efficient partitioning can be achieved by projecting data into the x-y plane and mapping them into a two dimension torus consisting of a discrete number of sections as shown in Figure 3.6. Let $\Delta\theta$ be the range of the angle in each segment S_n, where n is the

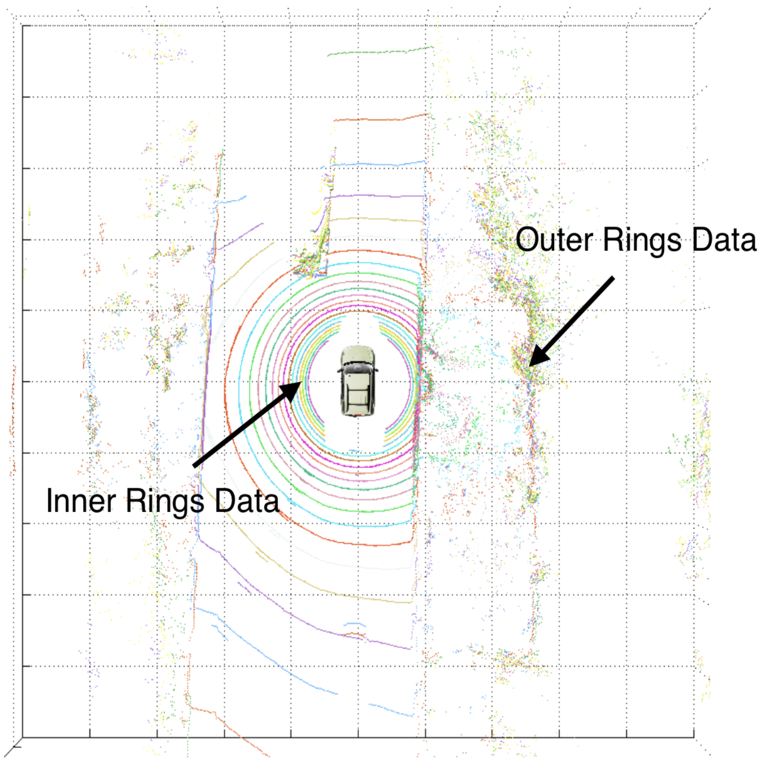


FIGURE 3.5: Points from inner rings versus from outer rings

index of segments and the total number of segment is $\frac{2\pi}{\Delta\theta}$. Then each segment is divided into several bins B_m^n, where m represent the index of bins in one segment. Because only the the inner rings of point cloud were to be processed, the minimum and maximum range search are decided by $r_{\min}$ and $r_{\max}$ and the the minimum and maximum width of each bin were also decided by $r_m^{\min}$ and $r_m^{\max}$ respectively. Let $\Delta r = r_m^{\max} - r_m^{\min}$ be the width of each bin so that the total number of bins in one segment is $\frac{r_{\max}-r_{\min}}{\Delta r}$.

In addition, let $\theta_n^{\min}$ and $\theta_n^{\max}$ be denoted the minimum and maximum horizontal angular position of each segment, where $\Delta\theta = \theta_n^{\max} - \theta_n^{\min}$ (e.g. $\Delta\theta = 36°$).

Therefore, a bin B_m^n by laser beam j at scanning position θ_k is given by:

$$\begin{aligned} & B_m^n = \{p_i | r_m^{\min} < \sqrt{(x_k^j)^2 + (y_k^j)^2} \leq r_m^{\max}, \\ & \theta_n^{\min} < \theta_k \leq \theta_n^{\max}\} \end{aligned} \tag{3.1}$$

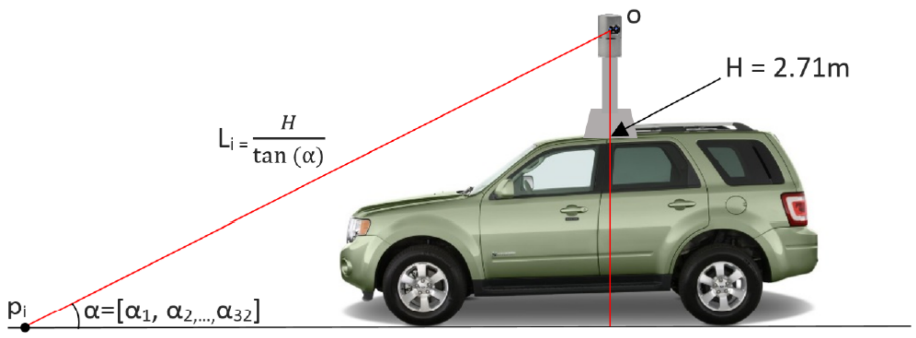
where $\theta_k = \mathrm{atan2}(y_k^j, x_k^j)$. Once all points have been mapped to a corresponding bin, a new set of points P_B in terms of bins is defined:

$$P_B = \{B_m^n | 0 < n \leq \frac{2\pi}{\Delta\theta}, 0 < m \leq \frac{r_{\max} - r_{\min}}{\Delta r}\} \tag{3.2}$$

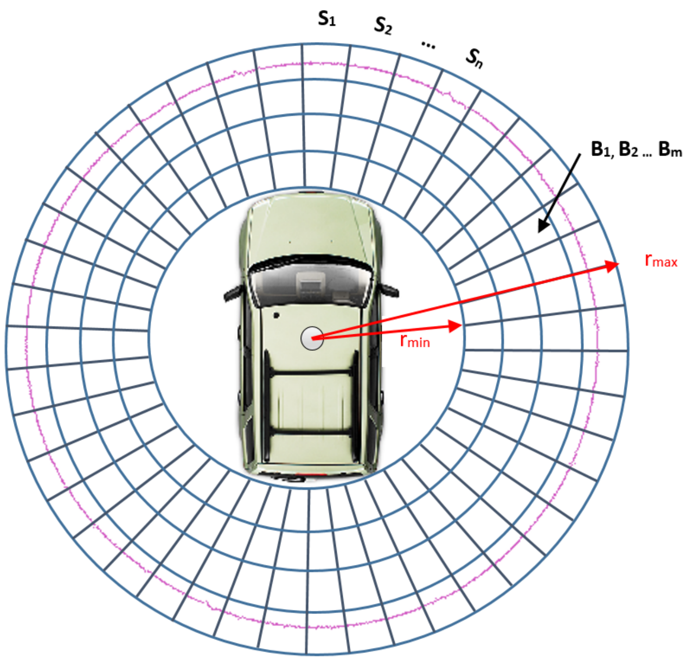
where m and n are the indexes of the bin. Since the $r_{\max}$ (e.g. 4m) and $r_{\min}$ (e.g. 1m) are the points which are not located inside the range. They are collected to be processed later.

3.3.2 Probability Occupancy Grid Map Modeling (OGM)

The basic idea of occupancy grid map is to represent the surrounding environment as an evenly spaced field of binary random variables indicating the presence of an obstacle at that location (See Figure 3.7). First of all, an occupancy grid map $M = \{B_m^n\}$ is defined, where n and m are indexes of a bin in the map, with each bin being one of two states: ground or non-ground.


O
H = 2.71m
$L_i = \frac{H}{\tan(\alpha)}$
p_i
$\alpha=[\alpha_1, \alpha_{2,...,}\alpha_{32}]$

(A)



(B)

FIGURE 3.6: Mapping of 3D LiDAR points to the bin

A standard approach is to average the height values of the points in each cell and define it as a part of "non-ground" if its feature value exceeds a predefined threshold. One of the advantages of the grid map is that it is robust to measurement noise [35].

In this section, a probability occupancy grid map is proposed and described as

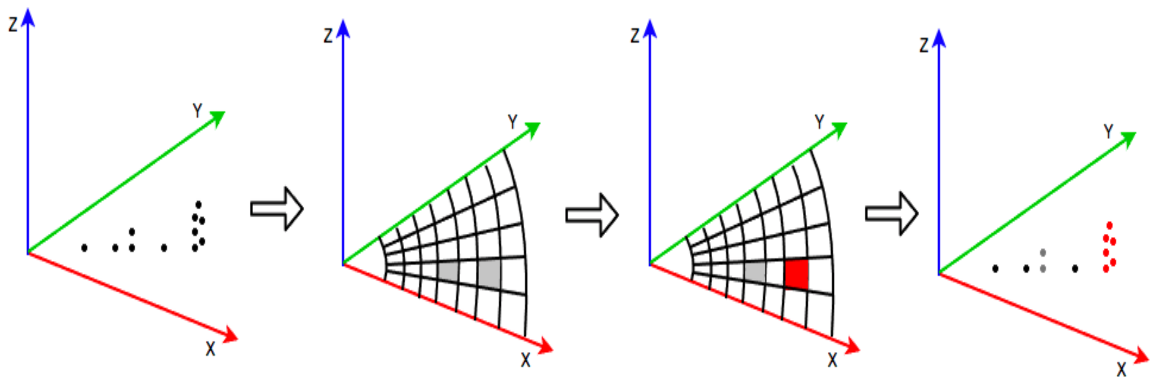


FIGURE 3.7: The process of OGM

shown in Figure 3.7. The goal of the probability occupancy grid map problem is to estimate the state of each bin B_m^n given the historical sensor measurements $z_{1..t}$ and the position of sensor $p_{1..t}$:

$$Pr(B_m^n|p_{1..t}, z_{1..t}) \tag{3.3}$$

Unfortunately, the dimensionality of the problem to be solved is huge since there are $2^{|M|}$ possible states (in the case of our LiDAR system, $M = 700$). But since the bins are independent of each other, the complexity can be reduced to $2|M|$. Because the events are independent of each other, the problem of estimating the states of all bins from the given map is decomposed into reconstructing the products of each bin's probability in terms of the marginal probability [36]:

$$Pr(M|p_{1..t}, z_{1..t}) = \prod Pr(B_m^n|p_{1..t}, z_{1..t}) \tag{3.4}$$

Then for each sensor reading, the occupancy probability is recursively estimated giving the measurements by using the Bayes' rule:

$$\begin{aligned} &Pr(B_m^n|p_{1..t}, z_{1..t}) \\ &= \frac{Pr(z_t|p_{1..t}, z_{1..t-1}, B_m^n)Pr(B_m^n|p_{1..t}, z_{1..t-1})}{Pr(z_t|p_{1..t}, z_{1..t-1})} \end{aligned} \tag{3.5}$$

From Markov assumption and Bayes' rule the above equation is further simplified as [37]:

$$\begin{aligned} Pr(B_m^n|p_{1..t}, z_{1..t-1}, B_m^n) &= Pr(z_t|p_t, B_m^n) \\ &= \frac{Pr(B_m^n|p_t, z_t)Pr(z_t|p_t)}{Pr(B_m^n|p_t)} \end{aligned} \tag{3.6}$$

And substituting into Eq.(3.5), then:

$$\begin{aligned} &Pr(B_m^n|p_{1..t}, z_{1..t}) \\ &= \frac{Pr(B_m^n|p_t, z_t)Pr(z_t|p_t)Pr(B_m^n|p_{1..t}, z_{1..t-1})}{Pr(B_m^n|p_t)Pr(z_t|p_{1..t}, z_{1..t-1})} \end{aligned} \tag{3.7}$$

where $Pr(B^n_m|p_t)$ is the prior probability of each occupancy cell and is independent of the current position $Pr(B^n_m|p_t) = Pr(B^n_m)$. Hence:

$$\begin{aligned} & Pr(B^n_m|p_{1..t}, z_{1..t}) \\ & = \frac{Pr(B^n_m|p_t, z_t)Pr(z_t|p_t)Pr(B^n_m|p_{1..t}, z_{1..t-1})}{Pr(B^n_m)Pr(z_t|p_{1..t}, z_{1..t-1})} \end{aligned} \tag{3.8}$$

The equation can be expanded once again for $Pr(z_t|p_{1..t}, z_{1..t-1})$ using Bayes' rule to get:

$$\begin{aligned} & Pr(B^n_m|p_{1..t}, z_{1..t-1}) \\ & = \frac{Pr(p_t|p_{1..t-1}, z_{1..t-1}, B^n_m)Pr(B^n_m|p_{1..t-1}, z_{1..t-1})}{Pr(p_t|p_{1..t-1}, z_{1..t-1})} \end{aligned} \tag{3.9}$$

Substituting into Eq.(3.8)

$$\begin{aligned} & Pr(B^n_m|p_{1..t}, z_{1..t}) \\ & = \frac{Pr(B^n_m|p_t, z_t)Pr(z_t|p_t)Pr(p_t|p_{1..t-1}, z_{1..t-1}, B^n_m)}{Pr(B^n_m)Pr(z_t|p_{1..t}, z_{1..t-1})} \times \\ & \frac{Pr(B^n_m|p_{1..t-1}, z_{1..t-1})}{Pr(p_t|p_{1..t-1}, z_{1..t-1})} \end{aligned} \tag{3.10}$$

Using the same proof, a matching update rule for the complementary state estimate $Pr(\bar{B}_m^n|p_{1..t}, z_{1..t})$ can be derived as:

$$\begin{aligned}
&Pr(\bar{B}_m^n|p_{1..t}, z_{1..t}) \\
&= \frac{Pr(\bar{B}_m^n|p_t, z_t)Pr(z_t|p_t)Pr(p_t|p_{1..t-1}, z_{1..t-1}, \bar{B}_m^n)}{Pr(\bar{B}_m^n)Pr(z_t|p_{1..t}, z_{1..t-1})} \times \\
&\frac{Pr(\bar{B}_m^n|p_{1..t-1}, z_{1..t-1})}{Pr(p_t|p_{1..t-1}, z_{1..t-1})}
\end{aligned} \tag{3.11}$$

Next, dividing Eq.(3.10) by Eq.(3.11):

$$\begin{aligned}
&\frac{Pr(B_m^n|p_{1..t}, z_{1..t})}{Pr(\bar{B}_m^n|p_{1..t}, z_{1..t})} \\
&= \frac{Pr(B_m^n|p_t, z_t)Pr(\bar{B}_m^n)Pr(p_t|p_{1..t-1}, z_{1..t-1}, B_m^n)}{Pr(\bar{B}_m^n|p_t, z_t)Pr(B_m^n)Pr(p_t|p_{1..t-1}, z_{1..t-1}, \bar{B}_m^n)} \times \\
&\frac{Pr(B_m^n|p_{1..t-1}, z_{1..t-1})}{Pr(\bar{B}_m^n|p_{1..t-1}, z_{1..t-1})}
\end{aligned} \tag{3.12}$$

Because the current sensor position p_t is known:

$$Pr(p_t|p_{1..t-1}, z_{1..t}, B_m^n) = Pr(p_t|p_{1..t-1}, z_{1..t-1}, \bar{B}_m^n) \tag{3.13}$$

Therefore, Eq.(3.12) is simplified to

$$\begin{aligned}&\frac{Pr(B^n_m|p_{1..t}, z_{1..t})}{Pr(\bar{B}^n_m|p_{1..t}, z_{1..t})}\\ &= \frac{Pr(B^n_m|p_t, z_t)Pr(\bar{B}^n_m)Pr(B^n_m|p_{1..t-1}, z_{1..t-1})}{Pr(\bar{B}^n_m|p_t, z_t)Pr(B^n_m)Pr(\bar{B}^n_m|p_{1..t-1}, z_{1..t-1})}\end{aligned} \tag{3.14}$$

As is common practice, the *log odds* of this probability can be computed instead of the probability itself [38]. Because Bayesian state update in *log odds* form using the *log odds* equation reduces the numerical errors from multiplying minuscule floating point numbers, as it assumes values from $-\infty$ until $+\infty$ instead of using just the interval between 0 to 1 where values close to 0 or 1 may cause numerical problem. Therefore, Eq.(3.14) can be transformed into:

$$\begin{aligned}&\log\frac{Pr(B^n_m|p_{1..t}, z_{1..t})}{Pr(\bar{B}^n_m|p_{1..t}, z_{1..t})} = \log\frac{Pr(B^n_m|p_t, z_t)}{Pr(\bar{B}^n_m|p_t, z_t)} + \log\frac{Pr(\bar{B}^n_m)}{Pr(B^n_m)}\\ &+ \log\frac{Pr(B^n_m|p_{1..t-1}, z_{1..t-1})}{Pr(\bar{B}^n_m|p_{1..t-1}, z_{1..t-1})}\end{aligned} \tag{3.15}$$

Let $L(x)$ be defined as:

$$L(x) = \log\frac{Pr(x)}{1 - Pr(x)} \tag{3.16}$$

Then Eq.(3.15) can be simplified as

$$L(B^n_m|p_{1..t}, z_{1..t}) = L(B^n_m|p_t, z_t) - L(B^n_m) + L(B^n_m|p_{1..t-1}, z_{1..t-1}) \tag{3.17}$$

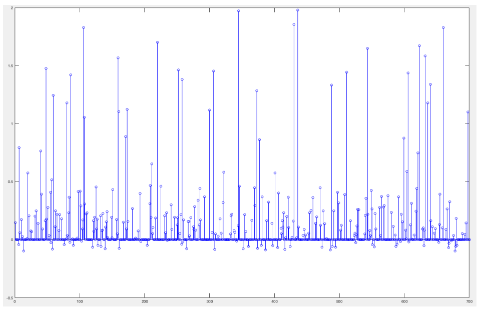
or in short

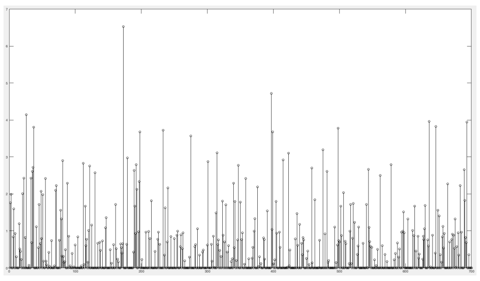
$$L_t = L(B^n_m|p_t, z_t) + L_{t-1} - L_0 \tag{3.18}$$

where $L(B^n_m|p_t, z_t)$ is called the inverse sensor model. The inverse sensor model specifies a distribution over the state variables as a function of measurement z_t. Considering a uniform prior probability which is $Pr(\bar{B}^n_m) = 0.5$. As a result, the term $L_0 = 0$. So the final formula is:

$$L_t = L(B^n_m|p_t, z_t) + L_{t-1} \tag{3.19}$$

As expressed in Eq.(3.19), the update function uses an inverse measurement model. Figure 3.8 demonstrates the attributes of each grid. To estimate the grid to be ground or non-ground, the mean height, the variance and the number of points are investigated for each grid. Let h^n_m denote the average height and μ^n_m denote the standard deviation of height in the current grid, N^n_m is the number of points of each bin.


(A) Average of height


(B) Standard deviation of height

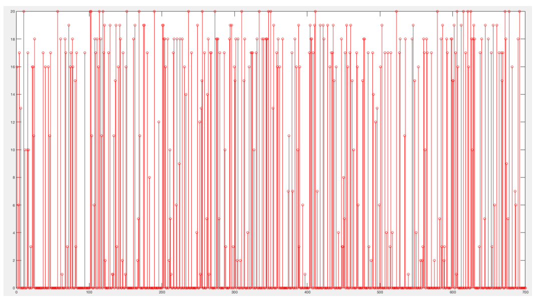

(C) Number of points in each grid

FIGURE 3.8: The attributes of grids

An function A_m^n is proposed here as a weighted sum of the partial evidences in terms of the *sigmoid function*.

$$\begin{aligned} A_m^n = &\frac{w_1}{1+e^{\alpha_1(h_m^n+\beta_1)}} + \frac{w_2}{1+e^{\alpha_2(\mu_m^n+\beta_2)}} \\ &+ w_3(1-\frac{1}{1+e^{\alpha_3(N_m^n+\beta_3)}}) \end{aligned} \tag{3.20}$$

where $w_1+w_2+w_3 = 1$, $\bar{\alpha} = \{\alpha_1, \alpha_2, \alpha_3\}$ and $\bar{\beta} = \{\beta_1, \beta_2, \beta_3\}$ are the predefined parameters respectively to adjust the sensitivity of the activation function. Based on real time experiments, the parameters are selected as $\bar{\alpha} = \{-6, -4, -4\}$ and $\bar{\beta} = \{-0.5, -2.5, -10\}$. Because the distribution of 3D point cloud is not uniform, the less the number of points in the grid, the harder to determine the grid to be ground or non-ground. So this problem is overcome by the third term of Eq.(3.20). The smaller of N_m^n, the higher of the value of A_m^n.

So the ternary occupancy indicator are defined as:

$$L(B_m^n|z_t) = \begin{cases} L_{ground} & A_m^n \leq 0.2 \\ L_{non-ground} & A_m^n \geq 0.8 \\ L_{unknown} & 0.2 < A_m^n < 0.8 \end{cases} \tag{3.21}$$

where the value of L_{ground}, $L_{unknown}$, and $L_{non-ground}$ is show at Table 3.1:

State	L_{ground}	$L_{unknown}$	$L_{non-ground}$
$L(B_m^n\|z_t)$	-1.39	0	1.39

Table 3.1: The occupancy indicator

In practice, the occupancy indicator of $L_{ground} = -1.39$ and $L_{non-ground} = 1.39$ are computed from the probabilities of $Pr = 0.8$ and $Pr = 0.2$ respectively according to Eq.(3.16).

One disadvantage of the update policy presented in Eq.(3.19) is that it requires to accumulate as many observations as possible before obtaining the current state. In order for the system to respond to dynamic changes in the environment immediately and overcome the overconfidence, the upper bound $L_{\max}$ and lower bound $L_{\min}$ is defined using a clamping update policy introduced by [39].

$$L_t = \max(\min(L(B_m^n|z_t) + L_{t-1}, L_{\max}), L_{\min}) \tag{3.22}$$

where the clamping values are $L_{\max} = 3.5$ and $L_{\min} = -2$ corresponding to maximum probability of 0.98 and minimum of 0.02. Therefore, the estimated posterior occupancy probability is computed in terms of Eq.(3.16):

$$Pr(B_m^n|p_{1:t}, z_{1:t}) = 1 - \frac{1}{1 + e^{L_t}} \tag{3.23}$$

The associated posterior occupancy probability value to conclude status of bin is as following:

$Pr(B_m^n\|p_{1:t}, z_{1:t})$	$0 \sim 0.3$	$0.3 \sim 0.6$	$0.6 \sim 1$
Status	Ground	Unknown	Non-ground

Table 3.2: The posterior probability and its associated status

3.3.3 Weighted Linear Regression

Once the ground points have been labeled by OGM, the next aim is to estimate the slope of the ground. Linear regression is used in this section which is widely used in statistical estimation for predicting the value of a dependent variable from an independent variable when the relationship between the variables can be described with a linear model [40].

Given data $\{(y_1, z_1), ..., (y_n, z_n)\}$, the problem is transformed to find the best fit straight line $z = \beta_1 + \beta_1 y$ such that the following equation is minimized:

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} [z_i - (\beta_0 + \beta_1 y_i)]^2 \tag{3.24}$$

Therefore, the objective is find the estimation of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the error by computing differentiation such that $\frac{\partial S}{\partial \beta_0} = 0$ and $\frac{\partial S}{\partial \beta_1} = 0$. Thus Eq.(3.24) is computed by rewriting the equations:

$$\begin{aligned} \Big(\sum_{i=1}^{n} y_i^2\Big)\hat{\beta}_1 + \Big(\sum_{i=1}^{n} y_i\Big)\hat{\beta}_0 = \sum_{i=1}^{n} y_i z_i \\ \Big(\sum_{i=1}^{n} y_i\Big)\hat{\beta}_1 + \Big(\sum_{i=1}^{n} 1\Big)\hat{\beta}_0 = \sum_{i=1}^{n} z_i \end{aligned} \tag{3.25}$$

In order to calculate for α and β, Eq.(3.26) can converted into the following matrix equation:

$$\begin{bmatrix} \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} y_i^2 \\ \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} y_i \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i z_i \\ \sum_{i=1}^{n} z_i \end{bmatrix} \tag{3.26}$$

Thus, the best fit line is obtained by solving above matrix by:

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} y_i^2 \\ \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} y_i \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{n} y_i z_i \\ \sum_{i=1}^{n} z_i \end{bmatrix} \tag{3.27}$$

Solving for the matrix yield the least squares parameter estimates:

$$\begin{aligned} \hat{\beta}_0 &= \frac{\sum_{i=1}^{n} y_i^2 \sum_{i=1}^{n} z_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} y_i z_i}{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2} \\ \hat{\beta}_1 &= \frac{n\sum_{i=1}^{n} y_i z_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} z_i}{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2} \end{aligned} \tag{3.28}$$

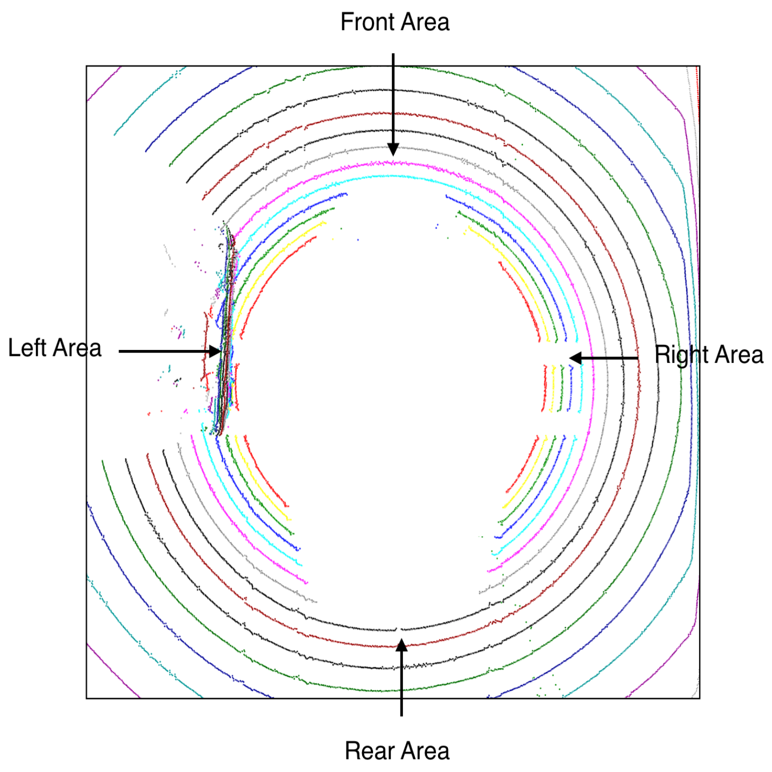


FIGURE 3.9: Front-Rear area vs Right-Left area

From the observation of the raw data in Figure 3.9, the point located near front-end area of the ego-vehicle has less noise than the right-left side area. In other words, the ground points may not be equally reliable. To address the reliability problem, one common solution for the weighted linear regression problem is called iteratively re-weighted linear squares (IRLS) [41]. As a result, Eq.(3.24) is modified to minimize the weighted sum of squares at time step t such that :

$$S_t(\beta_0, \beta_1) = \sum_{i=1}^{n} (w_i)_t[(z_i)_t - ((\beta_0)_t + (\beta_1)_t(y_i)_t)]^2 \tag{3.29}$$

Suppose the difference between the observed values and the corresponding fitted values in term of estimated $\hat{\beta}_0$ and $\hat{\beta}_1$ at time step t are the residuals:

$$(\hat{e_i})_t = (z_i)_t - (\hat{\beta}_0)_t - (\hat{\beta}_1)_t(y_i)_t \tag{3.30}$$

The solution of IRLS is to assigns a different weight to each case depending on the size of their residuals at time step t and thus minimizes the the weighted sum of squares until there's no "significant" change.

$$\min \sum_{i=1}^{n} (w_i)_t(e_i^2)_t \tag{3.31}$$

where $w_i \geq 0$ is the iterative weight of the i^{th} observation and is defined distance function by:

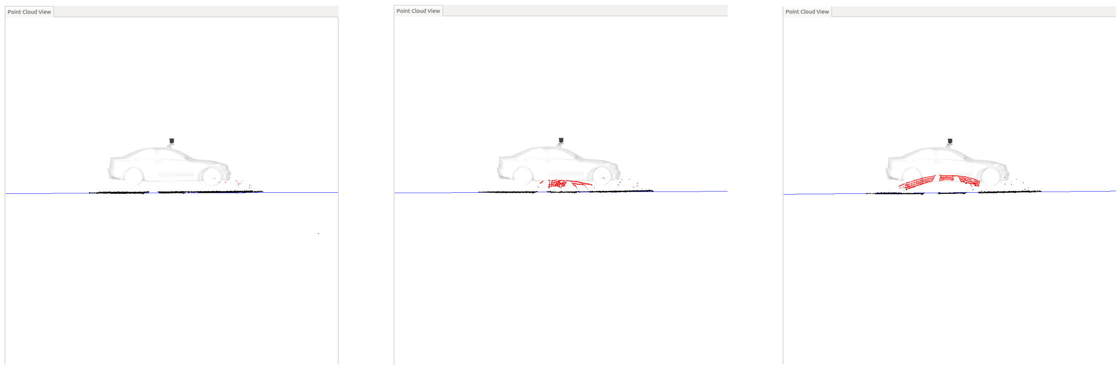
$$(w_i)_t = \frac{2}{(d_i^2)_t} \tag{3.32}$$

where d_i is a distance between the i^{th} point and the fit line at time step t. The calculated new $(\hat{\beta}_0)_{t+1}$ and $(\hat{\beta}_1)_{t+1}$ with respect to weights w_t are used to generate new weights w_{t+1}, which in turn are used in the next stage of IRLS until the convergence criterion is met.

Let δ and ϵ are the threshold values for convergence and tolerance of error respectively. Overall, the pseudo code of the IRLS implemented in our method is as following:

```
Data: Ground Points
Result: Estimated Slope Function
Initialization;
Compute weights for each point;
Project points into y-z coordinate system;
while the distance between the computed line and the points is greater than $\epsilon$ do
    Select random points;
    Set their weights to 1 ;
    Set other points to 0;
    Do RANSAC Algorithm;
    Do least squares weighted fit on these random picked points;
    while the difference between the computed solution and the previous one is greater than $\delta$ do
        Re-compute the weights for all points by Eq.(3.32) ;
        if all weights are 0 then
            set all weights to 1
        end
        Normalize the weights so that their sum is 1;
        Do least squares weighted fit for all points to the line;
    end
    if the last found solution is better than the current best solution then
        Save it as new best solution;
    end
end
Return the best solution;
```

In Figure 3.10, the result of slope estimation by our proposed method is demonstrated continuously robustness under different number of outliers. The red color represents the outliers while the blue color represent the slope estimation.



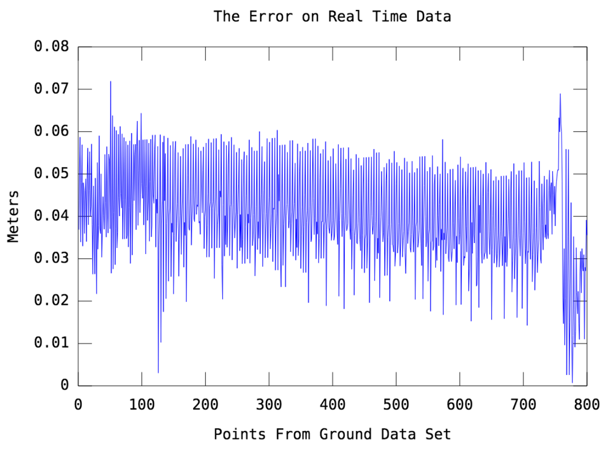
(A) Little number of outliers (B) Medium number of outliers (C) Large number of outliers

FIGURE 3.10: Slope estimation: Red represents the outliers, Blue represents the estimated slope

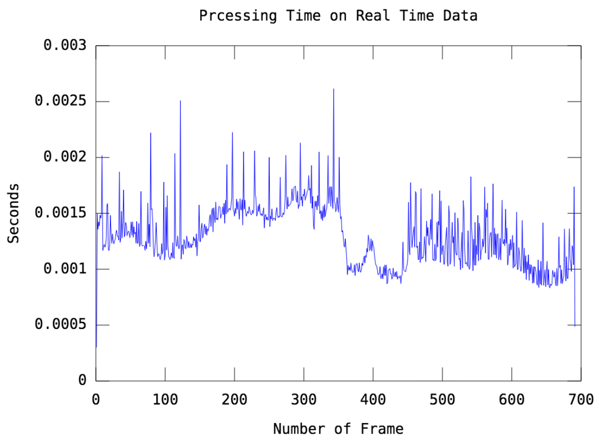
3.4 Experiment and Result

To validate this ground points identification strategy, experiments were conducted on our real time platform to verify the robustness and effectiveness of the proposed approach according to the scenarios of Table 2.3. The performance of each scenario was evaluated according to the following metrics: computation time, accuracy and robustness.

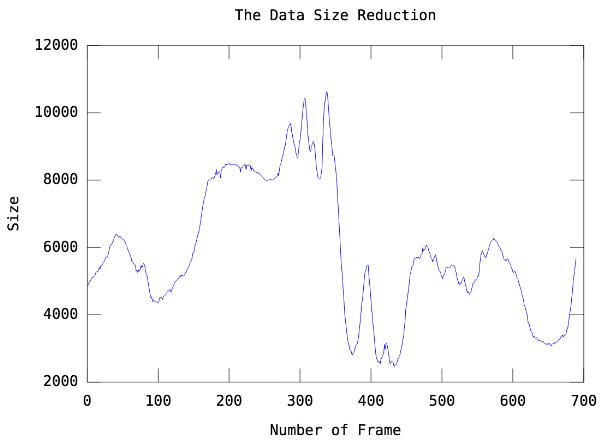
The developed algorithm is implemented in C++ to achieve real time capability, with additional speed-ups possible through the parallelization of the data acquisition modules. Figure 3.11 demonstrate the experimental result of the most challenged scenario: high traffic scenario. From Figure 3.11(A), the average processing time is about 1.3ms (769 Hz) which has significantly improved compared to [23, 31, 42]. In [23], the average processing time per frame was 250ms for ground segmentation. The fastest segmentation method proposed in [31], referred as GPINSAC, was 170ms. The real time 3D point cloud segmentation proposed in [42] cost about 150ms. Figure 3.11(C) shows the amount of non-ground data after processed by our proposed method with average of 5837 points while the input size of LiDAR raw data is about 55680 points at every frame. As a result, the number of data size to be processed has drop by 89.5% while retaining the most important information of raw data.



(A) The error between the estimated fit line and the ground points



(B) Real time computation time



(C) The post processing size of data

FIGURE 3.11: The experimental result of the most challenged scenario: high traffic

Table 3.3 illustrates the accuracy and robustness results. The data set is provided from our experiments contains manual labeled scans of different traffic scenes: high traffic scenario, medium traffic and low traffic. Each point has a label either ground or non-ground before preprocessing in order to compare our result to ground truth.

Traffic Condition	Light Traffic	Medium Traffic	High Traffic
Computation Time (ms)	0.536	1.22	1.30
# of Ground Points	21638	20127	20319
# of Non-Ground Points	5590	5707	5837
Precision	96.32%	94.6%	94.2%

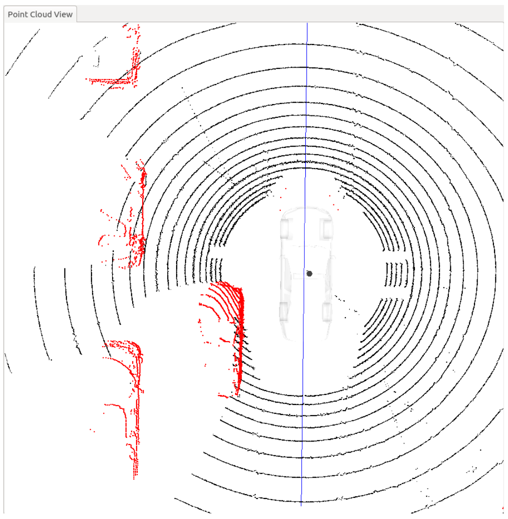
Table 3.3: Comparison in different scenarios

In pattern recognition with binary classification, precision, also known as positive predictive value, is defined as:

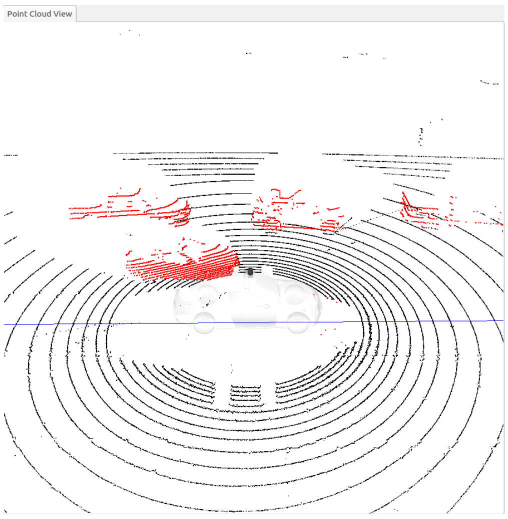
$$\text{Precision} = \frac{tp}{tp + fp} \tag{3.33}$$

where tp is the number of correctly labeled ground points and fp is the number of non-ground points mislabeled as ground points. Precision provide an insight in the performance of the proposed system. Low precision means that many non-ground points are classified as ground. From Table 3.3, the performance from the most challenged scenario: high traffic scenario has achieved a precision of 94.2%. This is because the data from the scans are collected in the express way where the terrains are usually very flat. As a result, the average processing time from both cases has significantly low computational complexity (1.02 ms in average) and the the average error from computed line and ground points is only approximately 0.04m as illustrated in Figure 3.11(B).

Additionally, Figure 3.12 shows the visualization of real time ground segmentation results by our proposed method in high traffic scenario. The black points represent the estimated ground points and the red points represent the non-ground points whereas the blue line illustrate the slope of the road.



(A) Apply estimated linear model for all points - top view



(B) Apply estimated linear model for all points - side view

FIGURE 3.12: Black represents ground points. Red represents non-ground points.

3.5 Conclusion

In this chapter, a novel real time occupancy grid map based ground detection approach has been presented. The approach can be divided into three phases. LiDAR raw points acquisition and preprocessing, occupancy grid mapping and weighted linear regression. Ground points are extracted in the first module while Cartesian coordinates representation is computed simultaneously for each single point. Through the first module, these points are separated into two set: dense data set and sparse data set. The dense data set fused in the occupancy grid map for building a probability map and weighted linear regression module for estimate the slope of the road. The complete segmentation of road area is done on the rest of sparse data set by applying regression model. Experiments are performed in real time in order to test the proposed method. From the experimental results, the proposed method on our real time LiDAR system demonstrates its robustness and efficiency under different scenarios. The extension based on our proposed system can be utilized to developed furthers algorithms for control or autonomous application to meet the real time criteria.

Chapter 4

The Algorithm: Design of Object Detection System

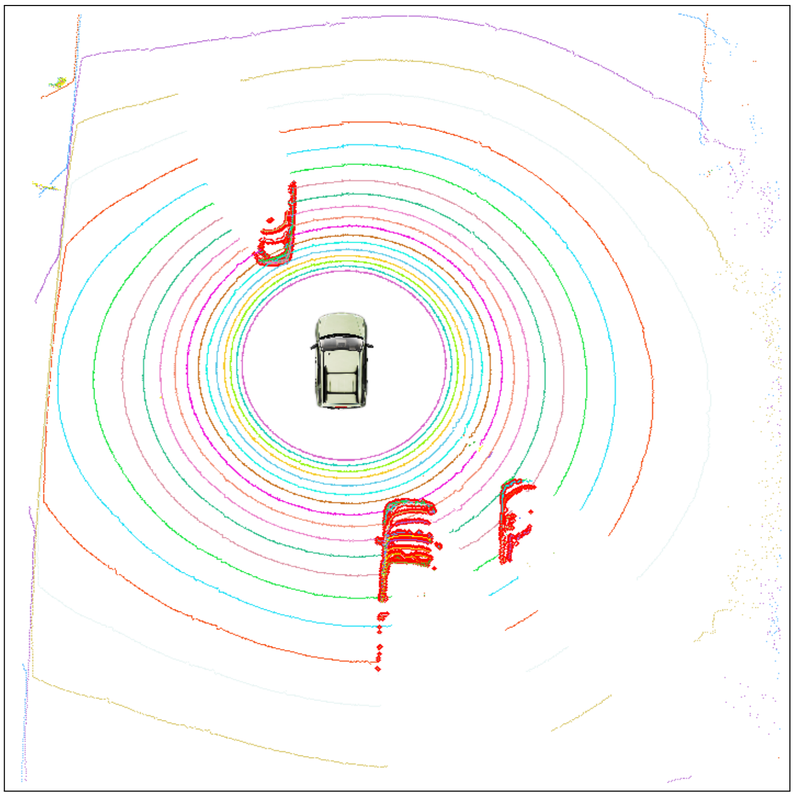


Figure 4.1: The result of object detection system

4.1 Introduction

This chapter present an introduction to clustering methods and describes our proposed clustering algorithm. Our proposed method use the static Kd-tree as our spatial data structure to manage nearest search operation due to its higher efficiency form experiments. Taking advantage of Kd-tree, a radially bounded nearest neighbor (RBNN) clustering method is proposed to segment the LiDAR points into different clusters. Our method is validated by its application to our real time LiDAR system and comparatively analysis. Figure 4.1 shows the overview of our developed object detection system in which the red color represents the object.

Object detection is a critical step in perception. Given the non-ground points from our ground detection system, the next step is to spatially cluster points with similar properties into single object. Point cloud clustering algorithm is a fundamental issue in processing LiDAR data and the quality of clustering algorithms largely determines the success of feature retrieval described later in Chapter 5. Most existing clustering methods have encountered two difficulties. First, project 3D range point cloud map into lower dimension leads to information loss. Second, process large volume of point cloud for real time application is a considerable computation complexity problem. In this chapter, we investigate the strategies of spatial data structure and develop LiDAR clustering methods.

4.2 Related Work

Most participants in DARPA Urban Challenge were projecting a three dimensional point cloud into lower dimension. For example, researchers in [26, 27] used an $2\frac{1}{2}$ occupancy grid to map the environment around a vehicle as a field of uniformly-distributed binary variables showing the status of grids (occupied or empty). One of the advantages is that the occupancy grid map allows for integration of several sensors into a unified representation, and the mapping strategy will become straight-forward.

Spatial data structures are hierarchical in nature, such as Kd-tree [43] and octrees [44]. Elseberg *et al.* [45] demonstrated different implementations of octree and Kd-tree for iterative closest points. Hornung *et al.* [46] presented a probabilistic and memory-efficient implementation of octree for mapping applications. However, Bentley *et al.* [47, 48] evaluated different methods of radius neighbor search and concluded that the Kd-tree is the most efficient and flexible spatial data structure for arbitrary dimensions. Jing Shen *et al.* [49] employed Kd-tree to manage the airborne LiDAR data after elimination of low and high outliers by using the elevation information.

The extracted dense data is clustered to identify target. A suitable solution can utilize spatial neighborhood relationships to compute the similarity between points. Shen *et al.* [50] leveraged a 3D Voronoi diagram to provides a reasonable description for the spatial topological neighborhood relationships. Other approaches try to cluster points by matching them to geometrical templates [51]. However, such model-based matching algorithms can only succeed if the point cloud is already reasonably segmented.

One of the difficulty for LiDAR data clustering is that no a priori knowledge on the distribution of points is available. Hence, classical supervised clustering methods are not useful. To overcome this problem, Frigui *et al.* [52] proposed a partitioning method which considers as much points as isolated clusters and then eliminates iteratively irrelevant clusters until reaching the correct number of clusters. Boley *et al.* [53] used Principal Component Analysis (PCA) to divide the point cloud which is also called Principal Direction Divisive Partitioning (PDDP). Frederick *et al.* [54] developed an algorithm to merge ellipsoids into larger ellipsoids by using a Minimum Spanning Tree (MST) algorithm. However, additional analysis is needed to recover relatively small holes which may be ignored in the model. Therefore, this method is not yet suitable for our application.

In [55], an algorithm is presented that efficiently segments a given 3D point cloud using a radially bounded nearest neighbor (RBNN) method while maintaining its ability for real time processing based on the static Kd-tree. The point cloud clustering technique is also based on our previous work in [24], which is a combination of RBNN and bounding box.

4.3 Spatial Data Structure

4.3.1 Kd-tree

A Kd-tree is a data structure used in computer science for organizing points in a k-dimensional space [43]. A Kd-tree is a generalization of a binary search tree that stores points in k-dimensional space. Each non-leaf node divides the space into two parts, known as subspaces. Points to the left represent the left subtree of that node and points to the right are represented by the right subtree. In other words, every level of a Kd-tree implicitly generates a splitting hyperplane. However, each node in the tree is associated with one of the k-dimensions, with the hyperplane perpendicular to that dimension's axis. Figure 4.2 below shows an example of Kd-tree in the 2-dimensional space. Red squares are dataset points, black lines are splits. The thinner the line is, the deeper is the node which corresponds to the split.

Kd-tree is very useful for searching nearest neighbor. Considering the number of dimensions k is fixed ($k = 3$), and dataset size is n, the complexity of the most important operations with Kd-tree is estimated:

- Building a Kd-tree has $O(n \log n)$ time complexity and $O(kn)$ space complexity
- Nearest neighbor search - close to $O(\log n)$
- m nearest neighbors - close to $O(m \log n)$

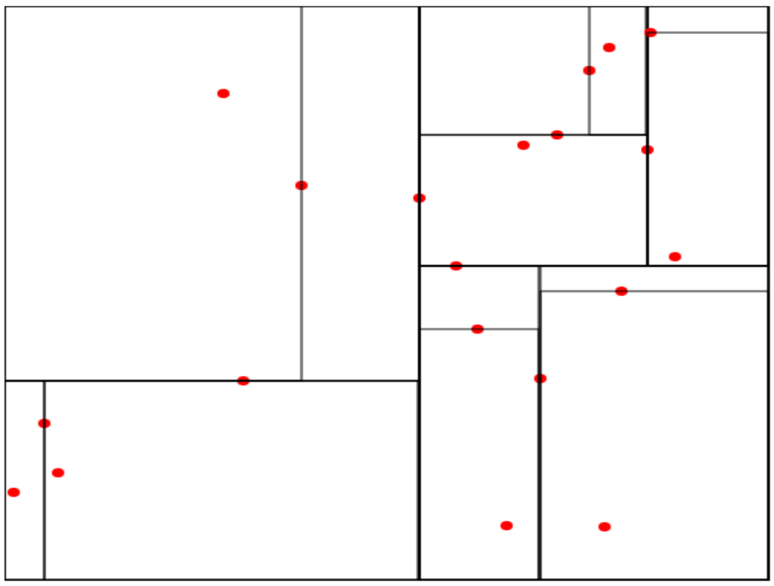


FIGURE 4.2: The Kd-tree in 2D space

It's easier to understand how a Kd-tree works by seeing an example. Figure 4.3 is a Kd-tree that stores points in a three-dimensional space.

Notice that in each level of the Kd-tree, if a pivot element such as x-axis is chosen, all points in the subtree with a smaller x value than the pivot will appear in the left subtree and all points with a larger x value will be in the right subtree. For instance, in Figure 4.3, the node (2, **3**, 7), the value of y-axis is shown in bold. Therefore, for all the nodes that are in its left subtree, their value of y-axis is less than 3.

The most efficient way to build a Kd-tree is to use a partitioning method to place the median point at the root and everything with a smaller one dimensional value to the left and larger to the right. Then repeat this procedure on both the left and the right of subtrees, until the last subtrees is only composed of one element. One of the advantage of Kd-tree is that it has been proved its usefulness

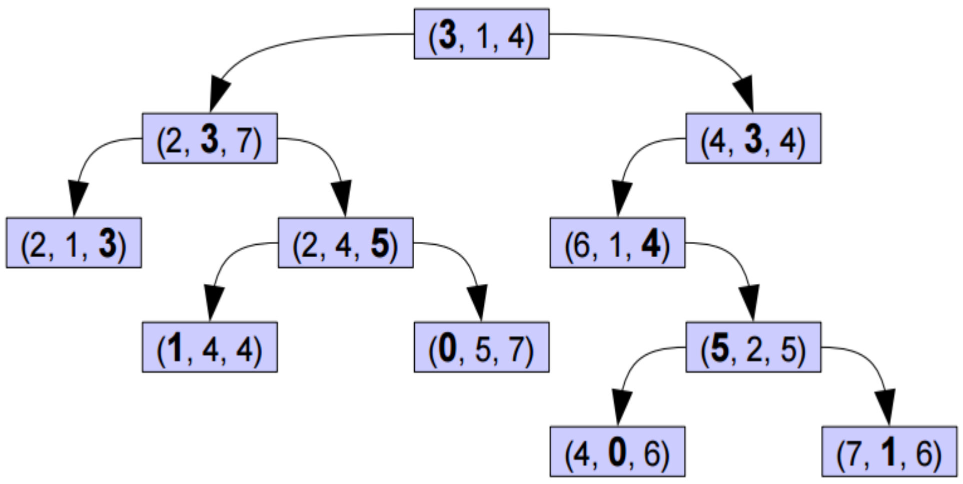


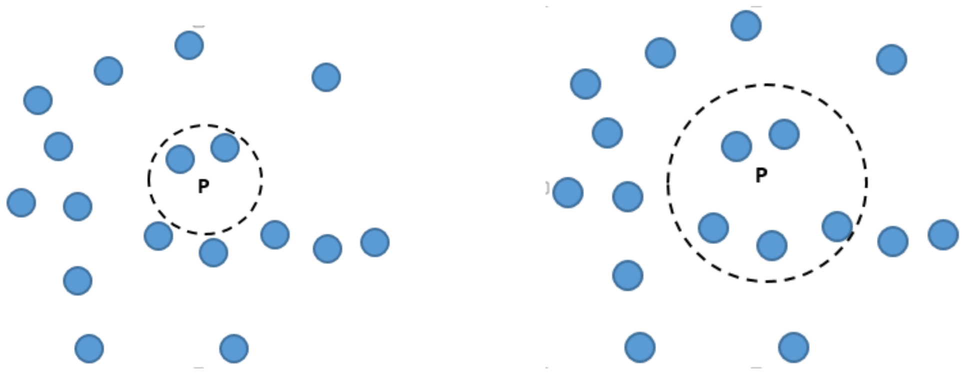
FIGURE 4.3: An 3D Kd-tree example

in the reduction of complexity of already existing three-dimensional models in an automatic and unsupervised way [56].

4.3.2 Nearest Neighbor Search in Kd-tree

The nearest neighbor (NN) search problem is a fundamental computational problem in computer vision, graphics, data mining, machine learning, and many other fields. Assume that the dataset is composed of N points $p_1, p_2, ..., p_n$. Given a query point p_q the NN query asks for the point $p_{nn} \neq p_q$ which is closer than any other point in the dataset. A more general form of the query is to ask for the k nearest points instead of just the closet one. So the k-NN query asks for the k points that are closest to p_q. The output of a k-NN query is a list of points sorted in increasing distance order from the query point.

Since the NN query retrieves answers according to the proximity of the points, a distance metric is required. The common used distance metric is the *Euclidean* *distance* [?]. Figure 4.4 shows examples of a 2-NN and a 5-NN queries with a fixed query point by using *Euclidean distance*.



(A) NN query for k=2 (B) NN query for k=5

FIGURE 4.4: An examples of 2-NN and 5-NN queries

The similarity between the range query and the NN query is apparent. In contrast, the maximum distance is defined to the query point, whereas the number of points that satisfy the criteria is not known in advance. For range search in Kd-tree, the tree is recursively descended depth-first, travel as far as possible from neighbor to neighbor within the maximum distance before backtracking. At the leave node, the distances of the point to the query vector are computed explicitly, and any point with a distance less than d is added to the list of results. The other side of children nodes are also being searched with the same procedure. The pseudo code of range search in Kd-tree is as follow:

```
Data: 3D Point Cloud
Result: Point Set
Initialization;
Query distance is $d$, start at root node;
$v, w$ be left and right children nodes;
if $v$ a leaf then
    report $cell(v) \cap d$;
    if $cell(v) \in d$ then
        report all points of $cell(v)$;
    end
    if $cell(v) \cap d = \emptyset$ then
        skip;
    end
end
else
    search subtree of $v$ recursively ;
end
do the same for $w$;
```

Algorithm 1: Range search in Kd-tree

4.4 Clustering Analysis

Clustering analysis has been an important area of research in the domain of computer science for data mining and patterns recognition in various kinds of data. This process can identify major patterns or trends without any supervisory information such as data labels. Broadly specified, the goal of clustering is to group a set of points into clusters such that points within the same cluster have high similarity to each other than to those in other clusters. In other words, it divides a set of objects into clusters each of which is a representative of a meaningful sub-population.

Before explaining the concepts of clustering algorithm, it is useful to formalize the clustering of point cloud data problem. Given n points in a k-dimensional space. The aim is to find m clusters $c_j, j = 1, 2, ..., m$ such that all clusters are disjoint $c_i \cap c_j = \emptyset, \forall i \neq j$ and each cluster has at least one point $\forall i = 1, 2, ..., m, c_i \neq \emptyset$. For the scope of LiDAR data, a 3-dimensional space is concerned in which the significant features of points are their spatial locations along the three Cartesian axes. So $\forall i = 1, 2, ..., n, p_i = \{x_i, y_i, z_i\}$, where x, y, z are the locations on three axes respectively. Assume the distance between two points p_i and p_j is $d_{ij} = ||p_i - p_j||_2$. The following Figure 4.5 shows the clustering of point cloud data. Three different color labels distinguished clusters and the grey line indicates the distance between every two points.

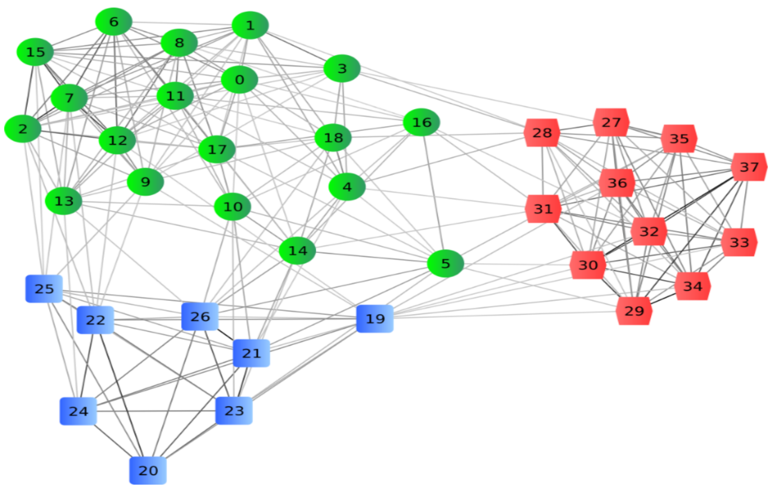
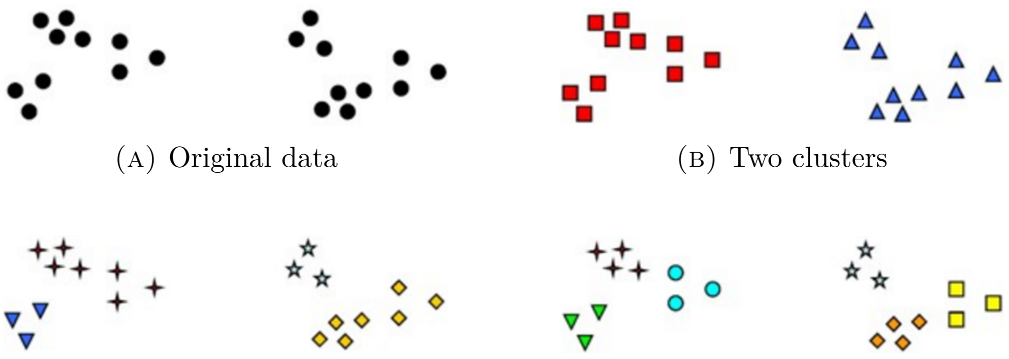


FIGURE 4.5: Clustering of points

In many applications, the notion of a cluster is not well defined. Figure 4.6 shows a set of points and three different ways of partition based on the number of clusters. The shapes of the markers represent cluster membership.


(A) Original data

(B) Two clusters

(C) Four clusters

(D) Six clusters

FIGURE 4.6: Different number of clusters for the same set of points

4.4.1 Review of Different Cluster Analysis Criteria

As a branch of statistics, data clustering has been studied for many years. Hierarchical and partitional are two such classes of clustering algorithms. Hierarchical clustering algorithms break up the data into a hierarchy of clusters. Paritional algorithms divide the data set into mutually disjoint partitions. Typically, partitional clustering is faster than hierarchical clustering. Therefore, The focus here will be on partitional clustering. A number of partitional clustering algorithms have been proposed to solve point cloud clustering problems. In this section, the three important techniques are used to introduce many of the concepts involved in cluster analysis, namely: K-mean, DBSCAN, and RBNN.

K-mean

K-mean is a simple partitional clustering technique that attempts to find a set of user-specified k clusters. These clusters are represented by their centroids. It has been used across a large range of application areas in different fields. The K-mean algorithm is to estimate the unknown cluster centers $M = \{m_1, ..., m_k\}$ based on the dataset $P = \{p_1, p_2, ..., p_n\}$. The K-mean algorithm aims to choose centroids that minimize the sum of squared criterion

$$\sum_{i=1}^{n} min(||p_i - m_i||^2) \tag{4.1}$$

where m_i is the closet cluster center to p_i. The most common algorithm, in Figure 4.7, uses an iterative refinement approach, following these steps:

1. Clusters the data into k clusters where k is predefined.
2. Select k points at random as cluster centers.
3. Assign points to their closest cluster center according to the *Euclidean distance* function.
4. Calculate the centroid or mean of all points in each cluster.
5. Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds.

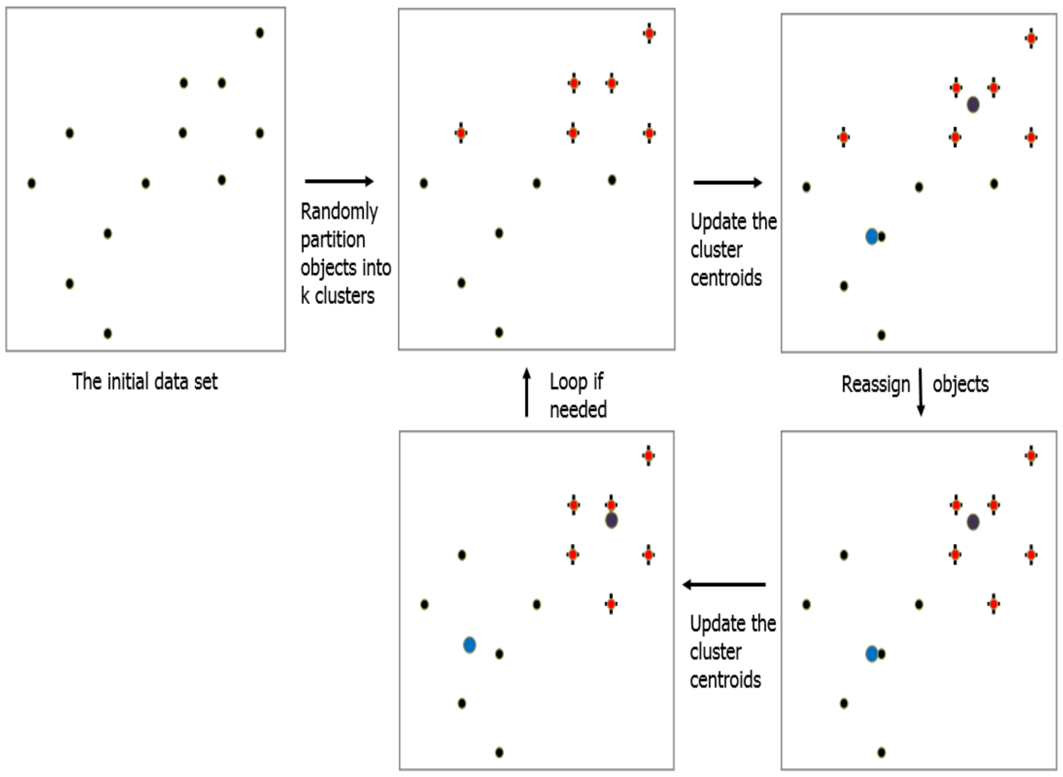

The initial data set
Randomly partition objects into k clusters
Update the cluster centroids
Reassign objects
Update the cluster centroids
Loop if needed

Figure 4.7: An example of K-mean clustering

These steps are repeated until a certain intra-cluster similarity objective function and inter-cluster dissimilarity objective function are optimized. Sensible initialization of centers is a very important factor in obtaining quality results from partitional clustering algorithms. Although K-mean is relatively efficient and robust when the data sets are distinct or well separated from each other, since in our application, the number of vehicles is unpredictable in dynamic environment, K-mean is not considered as clustering algorithm for our system.

DBSCAN

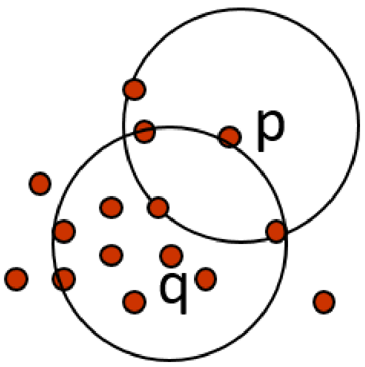
Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester *et al.* in 1996 [57]. It is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

Before describing the DBSCAN algorithm, some concepts must be explained. It uses the concept of **directly density reachability**, **density reachability** and **density connectivity** (See Figure 4.8). DBSCAN is formed by two parameters: ε, which specifies how close points should be to each other to be considered a part of a cluster; and **minPts** , which specifies how many neighbors a point should have to be included into a cluster.

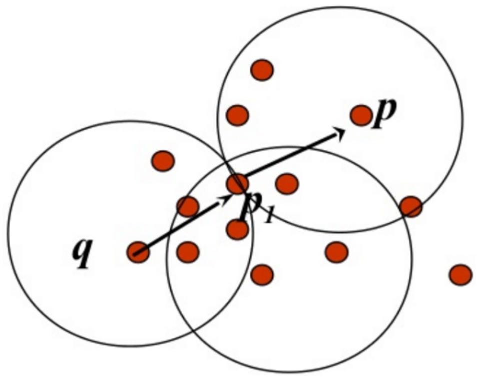
Directly Density Reachability - A point p_i is directly density reachable from a point p_j with respect to ε and minPts if the distance between p_i and p_j is less than ε.

Density Reachability - A point p_i is density reachable from a point p_j with respect to ε and minPts if there is a chain of point $p_1, p_2, ..., p_n$ and $p_1 = p_i, p_n = p_j$ such that p_{i+1} is directly density reachable from p_i.

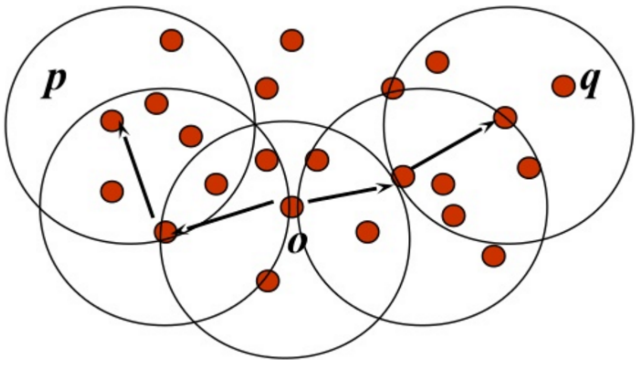
Density Connectivity - A point p_i and p_j are said to be density connected if there exist a point p_k such that both p_i and p_j are density reachable from p_k with respect to ε and minPts.



(A) Directly density reachability



(B) Density reachability



(C) Density connectivity

FIGURE 4.8: The concepts of DBSCAN

According to the above definition, a set of data points can be separate into two groups: the **border points** which are located on the extremities of the cluster, and the **core points**, which are located its inner region. The DBSCAN algorithm steps are described below:

1. Arbitrary select a point p_i.
2. Retrieve all points density reachable from p_i with respect to ε and **minPts**
3. A point p_i is a core point, if at least **minPts** points are within distance ε.

4. A point p_i is a border point, if no points are density reachable from p_i.

5. Continue the process until all of the points have been processed.

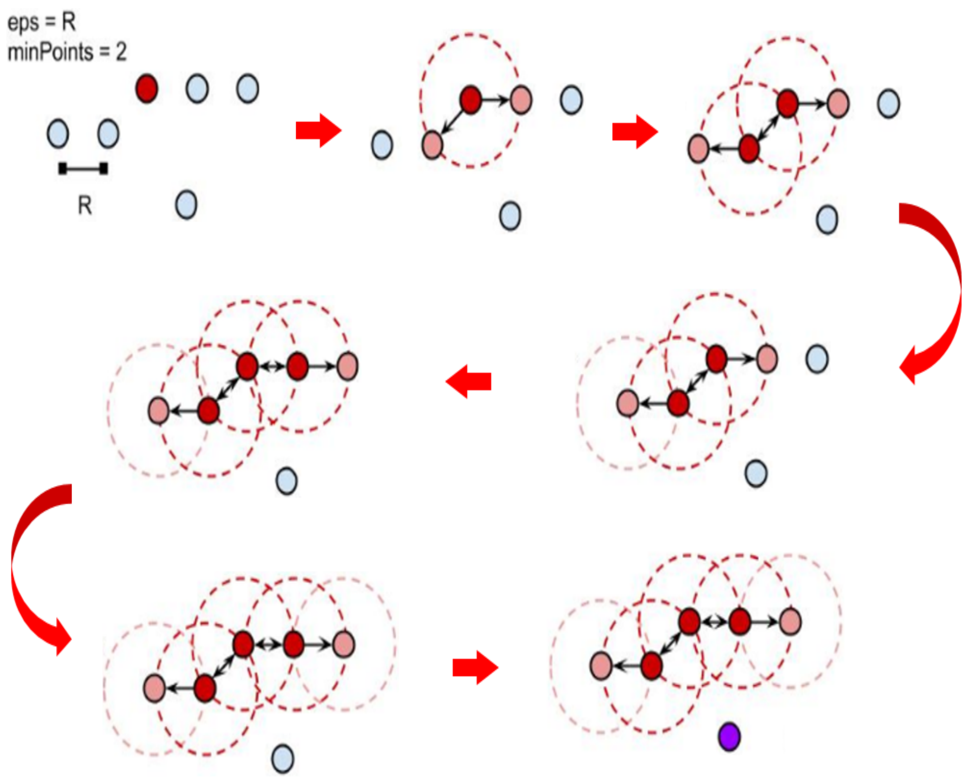


FIGURE 4.9: The steps of DBSCAN algorithm

Figure 5.9 provide a depiction of the DBSCAN algorithm in a step by step form. As can be seen from the last step image, there might be some points not belonging to any of the generated clusters, those points are outliers (noise).

RBNN

Matching 3D point clouds to geometrical shapes such as planes, cylinders or cube can only succeed if the point clouds are already reasonably segmented. For large

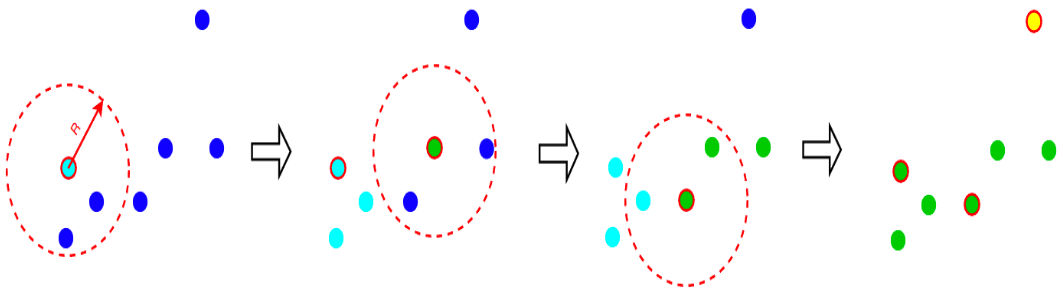


FIGURE 4.10: The steps of RBNN algorithm

3D point clouds obtained from complicated geometry, graph-based approaches are the most popular class of algorithm for robust and efficient segmentation of 3D laser data, because it can capture arbitrarily shaped clusters. Klaas *et al.* [55] presents an algorithm called radially bounded nearest neighbor (RBNN) by only using the concept of local neighborhood.

In RBNN every point in a cluster is connected to all neighbors that lie within a predefined radius r which is $\forall p_i, p_j, i \neq j, d_{ij} < r$. The main advantage of this method is that RBNN do not actually have to perform a nearest neighbor query for every point and consequently no graph cutting and rearranging of graph structures involved. So it is not even necessary to build a graph structure. This is in contrast to the K-mean which connects every point to its k nearest neighbors regardless of distance. Furthermore, RBNN also do not need initialization about the number of clusters which is one of the necessary conditions for evaluating in an unknown dynamic environment. Only two parameters, radius and the number of minimum points in one cluster, are needed to be initialized. The algorithm can be described by the following steps as shown in Figure 4.10:

1. Scan through all points from the data set.

2. For a current point which is not assigned to any cluster:
    - Search all neighboring points within a predefined radius.
    - If any of these neighbors are already assigned to a cluster, assign the current point to the same cluster.
    - If no neighboring points are assigned, then create a new cluster. Thus assign both current point and neighbors to the new cluster.
3. For a current point which is assigned to cluster:
    - Search all neighboring points within a predefined radius.
    - If there exists neighbor assigned to a different cluster, merge the cluster.
    - If no neighboring point is assigned, assign neighboring point to the cluster of the current point.

In RBNN, if neighbor points lies within a predefined radius, all the neighbor points are made to belonging to the same group. If the number of points in one cluster is less than our predefined minimum number, that cluster is considered an outlier.

4.5 RBNN Based Clustering using Kd-tree

The RBNN based clustering algorithm is nonparametric which means that the method does not need initialization. The pseudo code of our proposed clustering algorithm are shown below:

```
Data: 3D Point Cloud
Result: Cluster Set
Initialization;
Scan through each point;
if not end of set then
    Search neighbor points within radius;
    if current points has beed labeled then
        Ifneighbor point label different   Merge label and assign label to
          other unlabelled neighbor point;
        else
            Label other unlabelled neighbor point;
        end
    end
    else if neighbor point labelled then
        Assign same label to current point and other neighbor points;
    end
    else
        Create a new label and label current point and the nighbor points;
    end
end
else
    Scan through generated clusters;
    if Size of cluster within range then
        Keep current cluster;
    end
    else
        Discard current cluster;
    end
end
```

Algorithm 2: Pseudo code of our method

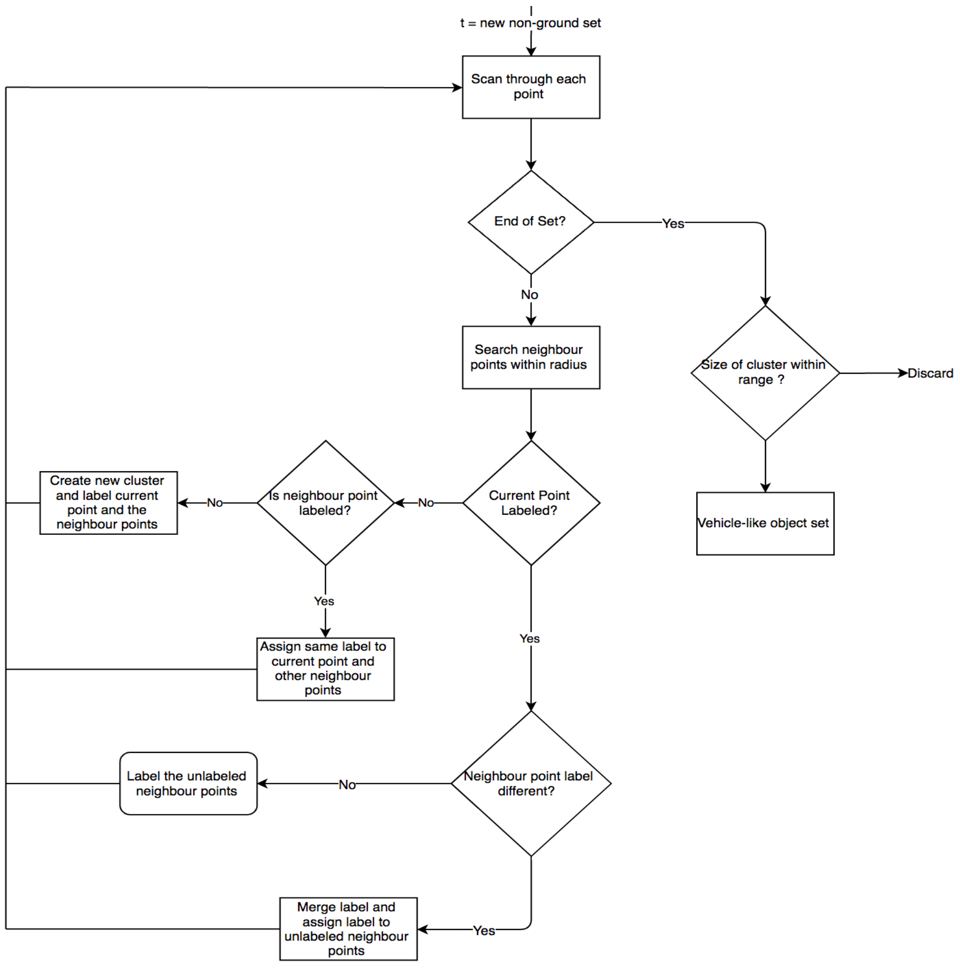

t = new non-ground set
Scan through each point
End of Set?
Yes
No
Search neighbour points within radius
Size of cluster within range ?
Discard
Vehicle-like object set
Current Point Labeled?
No
Is neighbour point labeled?
No
Create new cluster and label current point and the neighbour points
Yes
Assign same label to current point and other neighbour points
Yes
Neighbour point label different?
No
Label the unlabeled neighbour points
Yes
Merge label and assign label to unlabeled neighbour points

FIGURE 4.11: The flow diagram of clustering

The flowchart of the pseudo code is shown at Figure 4.11 which can be described as following:

1. Build a Kd-tree from the non-ground point cloud so that each point is represented by node.

2. Assign label to every node and initial the value of label as “not a number ” (NaN).

3. Iterate through each node and check for label.

4. If the label of current node is NaN:
   - Range search within predefined distance and return neighbor nodes.
   - If the value of label of any neighbor node is not NaN, copy the value of label to current node and other neighbor node.
   - If the value of label of all neighbor nodes are NaN, create new value and assign to all node from range search.
5. If the label of current node is not NaN:
   - Range search within predefined distance and return neighbor nodes.
   - If the value of label of all neighbor nodes are NaN, assign this label to all node from range search.
   - If the value of label of any neighbor nodes has different value, merge to the same value.
6. Project the clusters to 3D bounding box.
7. If the size of bounding box is over the threshold value:
   - Discard
8. Store the bounding boxes as vehicle-like cluster sets.

Figure 4.12 shows a bounding box is generated for each cluster in order for object features extraction as explained in the next chapter.

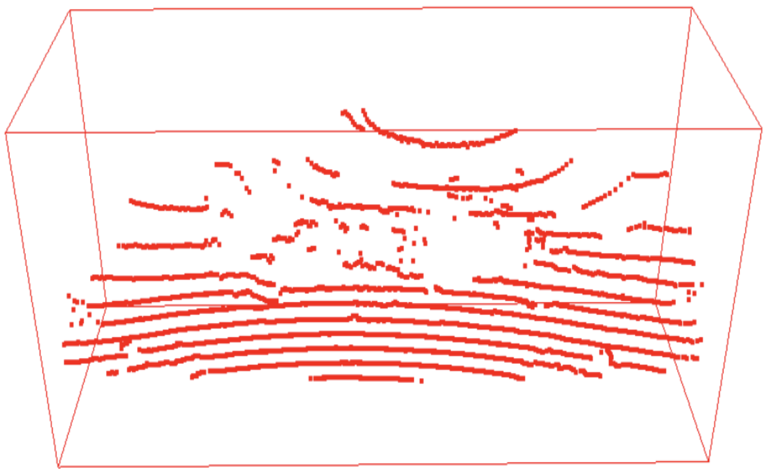
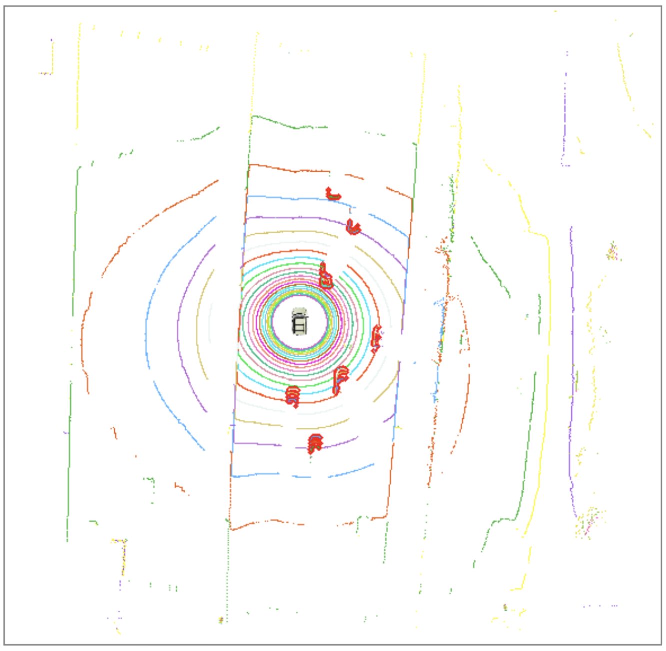


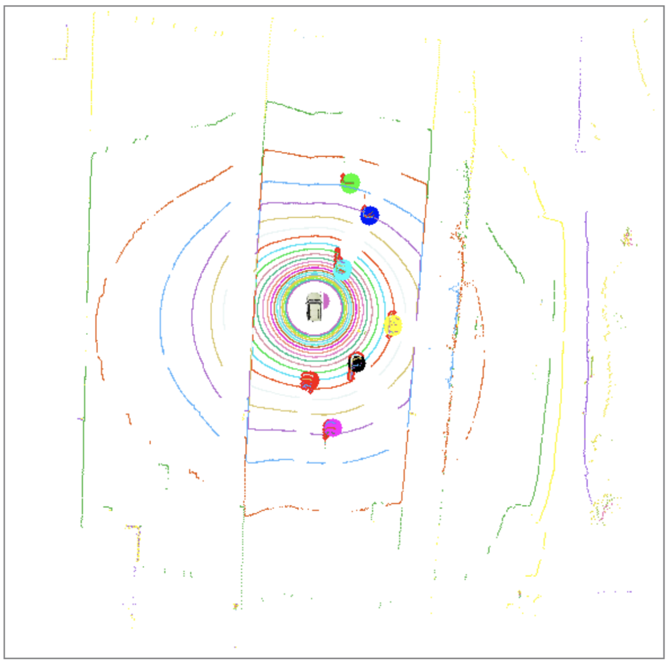
FIGURE 4.12: 3D bounding box around the cluster

4.5.1 Experiment and Result

To validate the proposed clustering strategy, experiments were conducted on our real time platform to verify the robustness and effectiveness of the proposed approach according to the scenarios of Table 2.3 from Chapter 2. Figure 4.13 shows the results of clustering algorithm from one frame. As can be seen from Figure 4.13(A), the set of non-ground data is rendered to color red. While Figure 4.13(B) demonstrates that seven distinct clusters with different colors are identified by our clustering algorithm.



(A) Object labeled by red color



(B) Clusters with different colors

Figure 4.13: An example of our methods on real time data

Evaluation clustering algorithms usually involves “internal" evaluation and “external" evaluation. Because the proposed algorithm is evaluated based on our real time data, so the candidate algorithm can not be compared with an existing “ground truth” classification. So the clustering analysis result is evaluated as “internal” evaluation.

The following methods can be used to assess the quality of clustering algorithms in terms of internal evaluation:

The Davies–Bouldin index (DBI) is computed by the following formula [58]:

$$\mathrm{DBI} = \frac{1}{n}\sum_{1}^{n} \max_{i \neq j} \frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \tag{4.2}$$

where n is the number of clusters, σ_i is the average distance of points in cluster to centroid c_i and $d(c_i, c_j)$ is the distance between centroid c_i and c_j. The DBI measures the average of similarity between each cluster. As the clusters should be compacted and separated, the lower DBI means better clustering result.

The Dunn index (DI) is given by [59]:

$$\mathrm{DI} = \frac{\min_{1 \leq i \leq j \leq n} d(c_i, c_j)}{\max_{1 \leq k \leq n} \delta_k} \tag{4.3}$$

where $d(c_i, c_j)$ represent the distance between the centroids and δ_k calculate the maximum distance between the farthest two points inside a cluster. The aim of DI is to identify sets of clusters that are compact and well separated. For a given assignment of clusters, a higher Dunn index indicates better clustering.

Table 4.1, Table 4.2 and Table 4.3 illustrate evaluation results. Since the main objective of our application is to detect vehicle, in practices, MinPts = 50 which is the minimum number of points to determine the type of cluster, and Radius = 1 are selected because the distance between moving vehicle is hardly below 1 meter.

Method	DBSCN	Our Method
Average DBI	1.042	1.0551
Average DI	1.49	1.231
Average Computation Time (s)	0.14587	0.06214
MinPts	50	50
Radius	1	1

Table 4.1: Evaluation Result: Scenario 1

Method	DBSCN	Our Method
Average DBI	1.0451	1.0546
Average DI	1.4112	1.233
Average Computation Time (s)	0.16123	0.07412
MinPts	50	50
Radius	1	1

Table 4.2: Evaluation Result: Scenario 2

Method	DBSCN	Our Method
Average DBI	1.041	1.06
Average DI	1.50	1.21
Average Computation Time (s)	0.03361	0.02932
MinPts	50	50
Radius	1	1

Table 4.3: Evaluation Result: Scenario 3

From the value of DBI and DI in above comparison table, although DBSCAN has a slightly better performance than RBNN, the computation cost of DBSCAN does not meet our real time criteria, whereas the processing time of RBNN is only 0.02932s in light traffic, 0.06214s in medium traffic, and 0.07412s in heavy traffic.

4.6 Conclusion

In this section, an RBNN based clustering algorithm is proposed by using static Kd-tree. The performance is evaluated by using LiDAR raw data sets under different traffic scenarios. The proposed algorithm was compared against the well known clustering technique DBDCAN. To effectively assess the performance of the proposed clustering algorithm, the Davies–Bouldin index (DBI) and the Dunn index (DI) which evaluate intra-cluster similarity and inter-cluster differences are computed. As illustrated by the results, the average values of both DBI and DI are robustness under different scenarios.

Chapter 5

The Algorithm: Design of Object Classification System

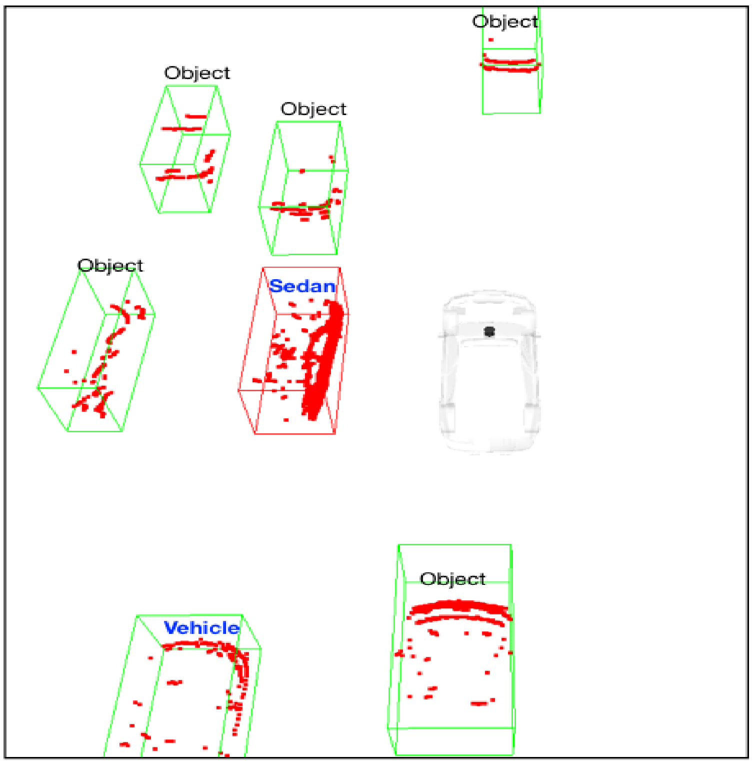


Figure 5.1: The result of object classification system

5.1 Introduction

In this chapter, a vehicle type classification approach is proposed using the convolutional neural network for the LiDAR system. In order to increase differentiation of information of vehicles and improve the performance accuracy, principle component analysis (PCA) is introduced to obtain the significant features of clusters by taking advantage of 3D data acquired from LiDAR. In addition, a support vector machine (SVM) is employed to identify the "Vehicle" cluster among other cluster data sets. Unlike conventional end-to-end classification methods, our method is able to discriminate type of vehicles in measured data. The data is collected and built from real time experiments which have included 5320 clusters with more than 2,660,000 points. The real time experimental results demonstrate the effectiveness and robustness of the proposed method. Figure 5.1 shows an example of the results of our proposed object classification system.

The development of a vehicle classification system is to help our LiDAR perception system to determine the geometry of its surrounding vehicles. The results from object detection system always include false clusters as shown in Figure 5.2 and its therefore necessary to recognize and classify the surrounding vehicles. To address this problem, real time vehicle classification techniques have been developed which aim to improve the accuracy of detection. In this chapter, an effective supervised learning approach is presented for vehicle classification on our real time LiDAR system.

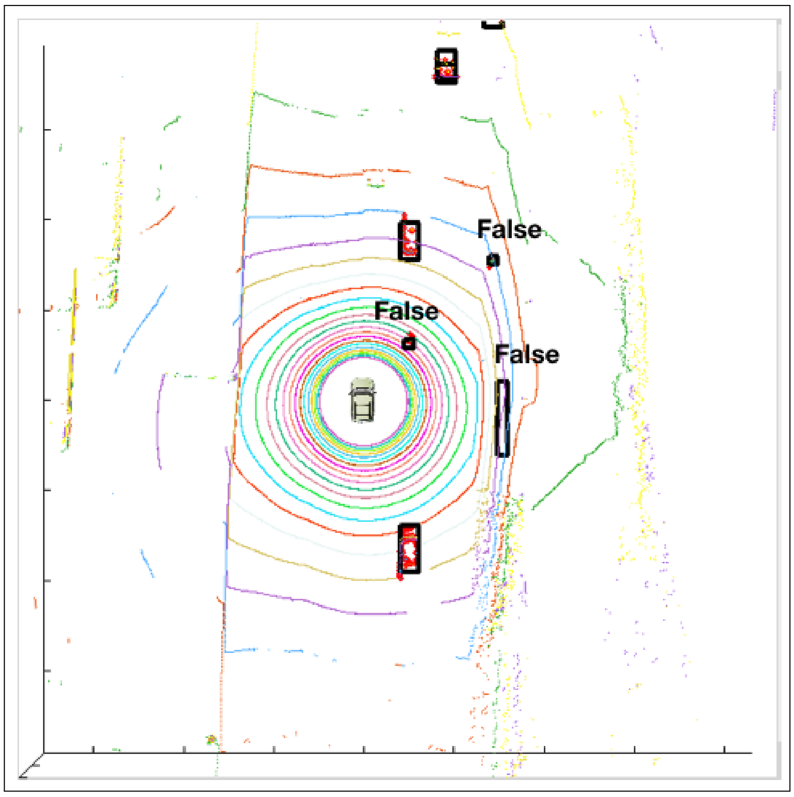


FIGURE 5.2: The result of object detection system

5.2 Related Work

Several machine learning approaches were proposed to solve vehicle classification problems. Amongst them, two typical methods have had a large impact on the research community: Support Vector Machine (SVM) and Artificial Neural Network (ANN). An SVM is a kernel based classification technique which was first introduced in 1995 by Cortes and Vapink [60]. This machine learning approach has become popular recently and has been applied to vehicle classification problems.

In a number of publications, SVM and PCA have been used to classify vehicle objects [66, 67]. Zhang *et al.* [61] implemented PCA-SVM to distinguish objects into trucks, passenger cars and vans. Although the result has achieved high accuracy, the processing time is not suitable for real time problems. Furthermore, most proposed strategies assume that there are to be no shadow conditions making them infeasible for application at night when shadow, and reflections are always present.

Zhiming *et al.* [62] leveraged SVM and PCA classifiers for vehicle detection. They divided measurements into training and testing set and performed vehicle recognitions using Scale Invariant Feature Transform (SIFT) method. Several other approaches utilizing PCA and SVM were mentioned in Zehang *et al.* [63]. However, SVM based approaches were designed for binary classification. How to effectively extend it for multiple classes classification is still an on-going research issue.

A number of artificial neural network (ANN) based technologies have been developed for vehicle classification. One of the most recent and very promising solutions is Convolutional Neural Networks (CNN). The CNN's strength is its ability to classify mutiple classes of objects. Krizhevsky *et al.* [64] applied CNN to classify the 1.2 million images into 1000 categories in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition. Other advantages of CNN are their relative robustness towards noise. Attila *et al.* [65] presented context based CNN for object classification such as for pedestrians and vehicles. Bo li *et al.* from Baidu research lab [33] used the CNN technique on 3D range scan data. They projected and discretized the range data into a 2D point maps and used a single 2D end-to-end Fully Convolution Network (FCN) to predict the confidence of vehicle and the bounding boxes simultaneously. A big disadvantage, is its long training time and the need for a large training data set that of their method would cover all classes of vehicles.

An efficient supervised learning approach is proposed for our real time LiDAR system. Our approach involves cascading a feature extractor PCA with a fixed kernel function SVM and a multiple layer perception CNN, referred as PCA-SVM-CNN. In comparison to individual classification algorithms such as SVM or CNN, experiments show that the proposed algorithm is able to deliver superior performance.

5.3 The Proposed Classification System

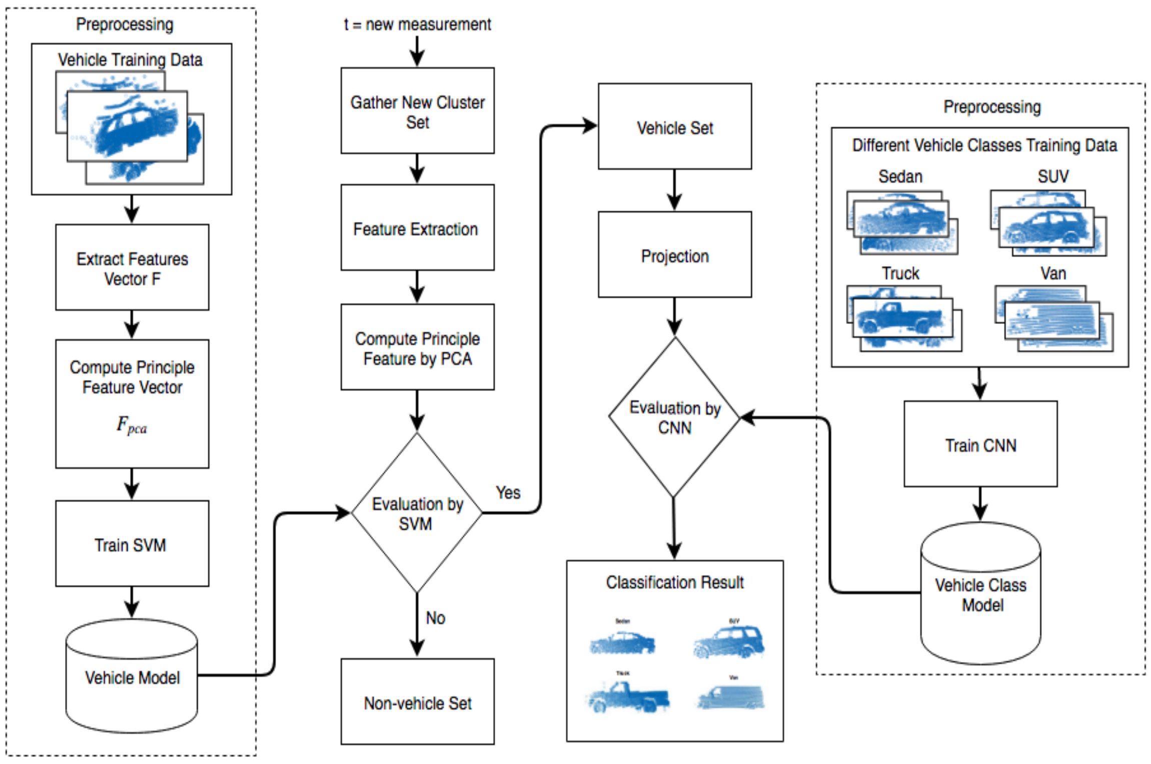


FIGURE 5.3: The data flow of our approach

Our approach to real time LiDAR classification system involves three main steps: (i) features selection, (ii) “vehicle” and “non-vehicle” classification, and (iii) vehicle types classification. In Figure 5.3, the proposed classification system, referred as PCA-SVM-CNN, is presented, which is described as following sections.

5.3.1 Feature Extraction by PCA

The use of PCA is to reduce dimensions and extract features from clusters [66]. It convert a set of observations into a set of values of linearly uncorrelated variables called principal components [68]. The first principal component is required to have the largest possible variance and the second component is computed under the constraint of being orthogonal to the first component and to have the second largest possible variance. The other components are computed likewise. Overall, PCA compresses data by reducing the number of dimensions without losing significant information and finds patterns in transformed feature space. The following are the steps to process LiDAR data by using PCA:

Suppose P is the input data set with n points and m features. p_i^j represent the i^{th} point and j^{th} feature. So P is represented by the following matrix

$$P = \begin{bmatrix} p_1^1 & p_1^2 & \dots & p_1^j \\ p_2^1 & p_2^2 & \dots & p_2^j \\ \vdots & \vdots & \dots & \vdots \\ p_i^1 & p_i^2 & \dots & p_i^j \end{bmatrix} \tag{5.1}$$

For example, given n points of LiDAR data set where each point is represented by Cartesian coordinate system, then P matrix is defined as

$$P = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \dots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{5.2}$$

The mean value of each dimension is $\mu = \{\mu_x, \mu_y, \mu_z\}$. Because the covariance is measured between 2 dimensions to see if there is a relationship between the 2 dimensions. For example, the formula of covariance matrix for variable x and y is computed by:

$$Cov(x, y) = \frac{\sum_{t=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{n - 1} \tag{5.3}$$

As the definition for the covariance matrix for a set of data with n dimensions is

$$Cov^{n \times n} = \{Cov_{ij} | Cov_{ij} = Cov(Dim_i, Dim_j)\} \tag{5.4}$$

where $Cov^{n \times n}$ is a matrix with n rows and n columns, Dim_i is the ith dimension. Because only 3 dimensional LiDAR data is processed in our application. Representing the covariance between dimensions as a matrix, e.g. for a 3 dimensional data set (x,y,z), is calculated by measuring the covariance between the x and y dimensions, the y and z dimensions, and the x and z dimensions. Therefore, the covariance matrix Cov has 3 rows and 3 columns:

$$Cov(x,y,z)^{3\times 3} = \begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix} \tag{5.5}$$

What PCA solves next step is the following eigenvalue problem of covariance matrix $Cov^{n\times n}$:

$$Cov^{n\times n} v_i = \lambda_i v_i \tag{5.6}$$

where λ_i are the eigenvalues and v_i are the corresponding eigenvectors. The eigenvalues is in order from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. To represent compressed vectors, the p eigenvectors corresponding to those p largest eigenvalues where $p \leq n$ is computed.

$$\begin{aligned} \Phi &= [v_1, v_2, ..., v_p] \\ \Lambda &= diag[\lambda_1, \lambda_2, ..., \lambda_p] \end{aligned} \tag{5.7}$$

Then

$$C\Phi = \Phi\Lambda \tag{5.8}$$

The parameter v denotes the approximation precision of the p's largest eigenvectors so that the following relation holds.

$$\frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq v \tag{5.9}$$

Note that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. Next a feature vector is formed which is a matrix of the selected eigenvectors.

Once the eigenvectors is selected, the feature vector is transposed and left multiplied it with the transpose of scaled version of original dataset. Thus the final matrix consisting of the principal components is shown below:

$$P_f = \Phi^T \times P^T \tag{5.10}$$

One of the main challenges of extracting meaningful features from point cloud for each bounding box is to find a compact representation that is differentiated from others as shown in Figure 5.4. In other words, compact representation can thereby dramatically reduce dimensionality of computation compared to the original data. Otherwise, the resulting classifier would not run in real time. On the other hand, discarding too much of the original information may cause the classifier to make a wrong decision.

Most studies use features exacted from the individual point of LiDAR data

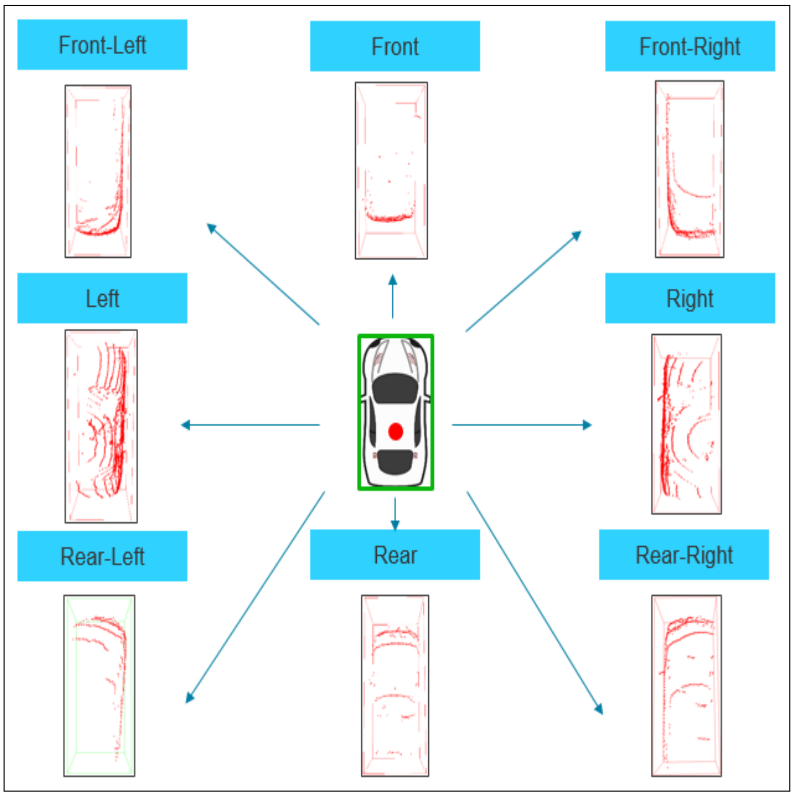


FIGURE 5.4: The features extraction from clusters

[42, 69, 70, 71] such as a histogram of point distribution or point positions. In our application, because the points have been clustered, the features are not restricted to "point" properties. Instead, some of the features should also be captured from "cluster" properties. For example, the sensor provide the intensity of laser reflection for the range of value from 0 to 255 according to [20]. Other approaches such as using 3D geometric features derived from the local covariance matrix [70, 71]. The point distribution features are described by the corresponding eigenvalues λ_1, λ_2, λ_3 computed from PCA to extract the features including dimensionality (linearity, planarity and sphericity). Therefore, both "point" and "cluster" features are extracted.

1. **The "cluster" Feature**:

   - Object volume (V), computed from object length (L), width(W) and height(H)

$$V = L \times W \times H \tag{5.11}$$

   - Number of points (N)
   - Number of laser ID (N_{id})
   - Density center ($P_d = \{x_d, y_d, z_d\}$)

$$P_d = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix} = \begin{bmatrix} \frac{\sum_{i=1}^{N} x_i}{N} \\ \frac{\sum_{i=1}^{N} y_i}{N} \\ \frac{\sum_{i=1}^{N} z_i}{N} \end{bmatrix} \tag{5.12}$$

2. **The "point" Feature**:

   - Maximum point intensity (I_{max})

$$I_{max} = f_{max}(\{I_1, I_2, ..., I_n\}) = \begin{cases} I_1 & \text{single sequence} \\ f_{max}(\{I_2, I_3, ..., I_n\}) & I_1 \leq I_2 \\ f_{max}(\{I_1, I_3, ..., I_n\}) & otherwise \end{cases} \tag{5.13}$$

- Minimum point intensity (I_{min})

$$I_{min} = f_{min}(\{I_1, I_2, ..., I_n\}) = \begin{cases} I_1 & \text{single sequence} \\ f_{min}(\{I_2, I_3, ..., I_n\}) & I_1 \geq I_2 \\ f_{min}(\{I_1, I_3, ..., I_n\}) & otherwise \end{cases} \tag{5.14}$$

- Mean point intensity (I_{mean})

$$I_{mean} = \sum_{i=1}^{N} \frac{I_i}{N} \tag{5.15}$$

- Variance point intensity ($Var(I)$)

$$Var(I) = \frac{\sum_{i=1}^{N}(I_i - I_{mean})^2}{N} \tag{5.16}$$

- Height variance of point ($Var(H)$)

$$Var(H) = \frac{\sum_{i=1}^{N}(H_i - H_{mean})^2}{N} \tag{5.17}$$

- Distance variance of point ($Var(d)$)

$$Var(d) = \frac{\sum_{i=1}^{N}(d_i - d_{mean})^2}{N} \tag{5.18}$$

- Sphericity (S_λ); Linearity (L_λ); Planarity (P_λ)

  (a) $Cov(P) = \frac{\sum_{i=1}^{N}(p_i - \bar{p})(p_i - \bar{p})^T}{N}$

  (b) Compute eigenvalues $\vec{\lambda} = \{\lambda_1, \lambda_2, \lambda_3\}$ by $Cov(P)\vec{v} = \vec{\lambda}\vec{v}$

  (c) $L_\lambda = \lambda_1 - \lambda_2; P_\lambda = \lambda_2 - \lambda_3; S_\lambda = \lambda_3$

Thus the feature vector is defined as:

$$\begin{aligned}\vec{F} =&(V, N, N_{id}, P_d, I_{max}, I_{min}, I_{mean},\\ &Var(I), Var(H), Var(d), L_\lambda, P_\lambda, S_\lambda)\end{aligned} \tag{5.19}$$

Because the feature vector $\vec{F}$ has dimension of thirteen. PCA is utilized to reduce the dimensions required for classifying new data and to produces a set of principal components, which are orthonormal eigenvalue/eigenvector pairs. This reduces the dimensionality of our data by restricting attention to those directions in the feature space in which the variance is greatest. PCA is applied on the original feature vector set of the data. Based on our experiments for the trade-off between accuracy and computation complexity, the most four significant principle components is selected so that the feature vector of the i^{th} cluster is: $\vec{F_i} = (F_i^1, F_i^2, F_i^3, F_i^4)$. The new feature vector set F_{pca} is thus formed by $F_{pca} = \{\vec{F_i} | 0 < i \leq n\}$, where i is the index of cluster and n is the number of clusters.

Therefore, the selected feature vector of the object is evaluated by SVM classifier in terms of the corresponding category based on the object's location. So each detected object could be assigned one of the labels "vehicle" or "non-vehicle".

5.3.2 Binary Classification by SVM

In this section, SVM is briefly described for binary classification problems as shown in Figure 5.5. The output of SVM is a vehicle cluster set which will be classified into different types of vehicles by the CNN as showed in the next section.

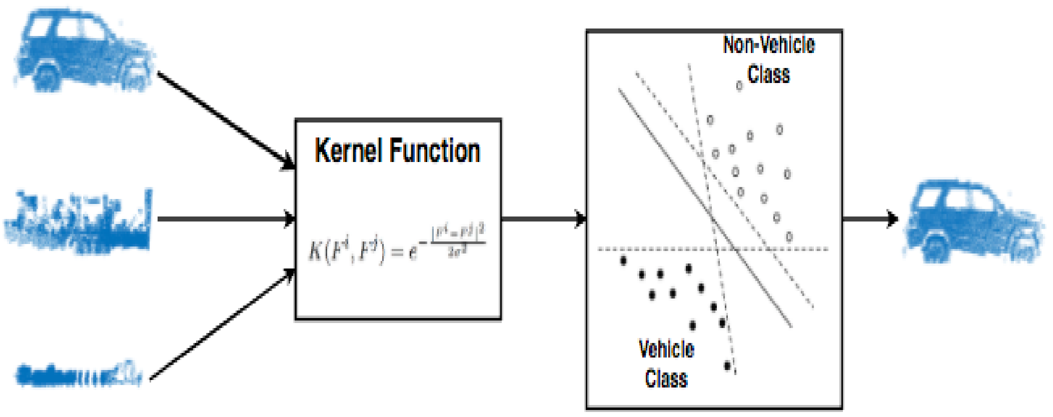


FIGURE 5.5: The data flow of binary SVM

Let the label of i^{th} cluster y_i be one of two values such that

$$y_i = \begin{cases} +1 & \text{vehicle} \\ -1 & \text{non-vehicle} \end{cases} \tag{5.20}$$

Then there exists a hyperplane that separates the positive from the negative training examples such that

$$\begin{aligned} \vec{F_i} \cdot \vec{w} + b \geq 0 \quad \text{when} \quad y_i = +1 \\ \vec{F_i} \cdot \vec{w} + b \leq 0 \quad \text{when} \quad y_i = -1 \end{aligned} \tag{5.21}$$

where $\vec{w}$ is the normal to the hyperplane and b is the perpendicular distance of the hyperplane to the origin, this gives the function

$$f(\vec{F_i}) = \vec{F_i} \cdot \vec{w} + b \tag{5.22}$$

$f(\vec{F_i})$ can be interpreted as the functional distance of an instance from the hyperplane. For $f(\vec{F_i}) < 0$ the point would be classified as "non-vehicle", and "vehicle" if $f(\vec{F_i}) > 0$.

To solve this, *Canonical hyperplane* [73] is defined which separates the data from the hyperplane by a "distanc" of at least 1. That is, considering the equation that satisfy:

$$\begin{aligned} \vec{F_i} \cdot \vec{w} + b \geq +1 \quad \text{when} \quad y_i = +1 \\ \vec{F_i} \cdot \vec{w} + b \leq -1 \quad \text{when} \quad y_i = -1 \end{aligned} \tag{5.23}$$

or more compactly:

$$y_i \cdot (\vec{F_i} \cdot \vec{w} + b) \geq +1 \tag{5.24}$$

Let a feature vector in such hyperplane is represented by a data point. Thus the data points for which the equality condition in Eq.(5.24) holds are called the support vectors showed in Figure 5.5 by extra circles.

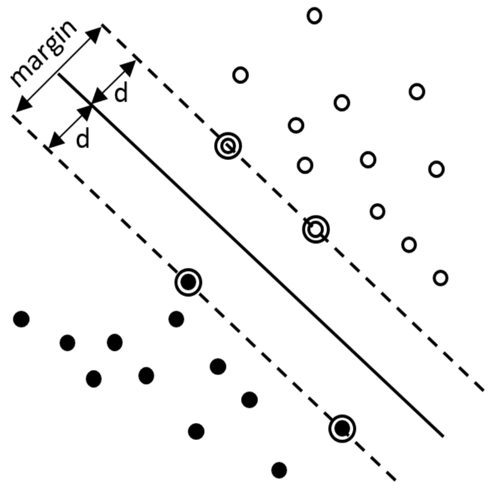


FIGURE 5.6: Optimal separating hyperplane with maximum margin

To obtain the geometric distance d from the hyperplane to a given data point, $\vec{w}$ is normalized by the magnitude so that the distance of i^{th} data point:

$$d(\vec{w}, b, \vec{F_i}) = \frac{y_i \cdot (\vec{F_i} \cdot \vec{w} + b)}{||\vec{w}||} \geq \frac{1}{||\vec{w}||} \tag{5.25}$$

From the Eq.(5.25), the optimal hyperplane is given by maximizing the margin, ρ, subject to the constraints of Eq.(5.24). The margin is given by

$$\rho = \frac{2}{||\vec{w}||} \tag{5.26}$$

Hence the hyperplane that optimally separates the data is accomplished by minimizing $\frac{1}{2}||\vec{w}||^2$, subject to the constraints of Eq.(5.24). The solution to this optimization problem is given by introducing *Lagrange multiplier* [72]. The Lagrangian is formed by multiplying the constraints by the positive *Lagrange multiplier* $\vec{\beta} = \{\beta_1, \beta_2, ..., \beta_k\}$, where k is the number of solutions that Eq.(5.25) holds. This gives the following Lagrangian:

$$L(\vec{w}, b, \vec{\beta}) = \frac{1}{2}\vec{w}^T\vec{w} - \sum_{i=1}^{k} \beta_i[\vec{F_i} \cdot \vec{w} + b - 1] \tag{5.27}$$

The Langragian L has to be minimized with respect to the primal variable $\vec{w}$ and b. The solution for our primal problem is a differentiated L with respect to $\vec{w}$ and b is given by:

$$\begin{aligned} \frac{\partial L}{\partial \vec{w}}(\vec{w}, b, \vec{\beta}) = 0 \\ \frac{\partial L}{\partial b}(\vec{w}, b, \vec{\beta}) = 0 \end{aligned} \tag{5.28}$$

Solving the Eq.(5.28) leads to:

$$
\begin{aligned}
&\vec{w} = \sum_{i=1}^{k} \beta_i y_i \vec{F}_i \\
&\sum_{i=1}^{k} \beta_i y_i = 0
\end{aligned}
\tag{5.29}
$$

Substituting Eq.(5.29) into Eq.(5.27), so the problem is eventually transformed into the dual problem which eliminates the variables $\vec{w}$ and b:

$$
\begin{aligned}
\text{minimize:} \quad & L(\vec{\beta}) = -\sum_{i=1}^{k} \beta_i + \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} y_i y_j \beta_i \beta_j \vec{F}_i \vec{F}_j \\
\text{subject to:} \quad & \sum_{i=1}^{k} y_i \beta_i = 0, \forall i, \beta_i \geq 0
\end{aligned}
\tag{5.30}
$$

So given a training set, L has to be minimized by finding the optimal *Lagrange multipliers* in $\vec{\beta}^*$. Once the optimal solution $\vec{\beta}^*$ is obtained, the optimal variable $\vec{w}^*$ and b^* is calculated by:

$$
\begin{aligned}
&\vec{w}^* = \sum_{i=1}^{k} \vec{\beta}^* y_i \vec{F}_i \\
&b^* = -\frac{1}{2}(\vec{w}^* \cdot x^+ + \vec{w}^* \cdot x^-)
\end{aligned}
\tag{5.31}
$$

where x^+ and x^- are obtained by taking any “positive" or “negative" support vector that satisfying:

$$\begin{aligned}(\vec{w}^* \cdot x^+ + b^*) &= +1 \\ (\vec{w}^* \cdot x^- + b^*) &= -1\end{aligned} \tag{5.32}$$

The decision function of an input feature vector $\vec{F}_{\text{new}}$ is then determined by:

$$\begin{aligned}f_d(\vec{F}_{\text{new}}) &= \vec{F}_{\text{new}} \cdot \vec{w}^* + b^* \\ &= \sum_{i=1}^{k} \vec{\beta}^* y_i \vec{F}_i^T \vec{F}_{\text{new}} + b^* \\ &= \begin{cases} 1 \quad (\text{“vehicle”}) \quad \text{if} \quad f_d(\vec{F}_{\text{new}}) \geq 0 \\ -1 \quad (\text{“non-vehicle”}) \quad \text{if} \quad f_d(\vec{F}_{\text{new}}) < 0 \end{cases}\end{aligned} \tag{5.33}$$

5.3.3 Kernel Functions

For non-linear SVM, finding a linear separating hyperplane in a multi-dimensional space solves the non-linear classifier problem. Suppose that an input feature vector $\vec{F}$ is mapped into feature space $\mathbb{F}$ with the mapping function Φ. Figure 5.7 illustrate the objective of the transformation to feature space. A problem that is not linearly separable in the input space can be linearly separable by a transformation to a feature space.

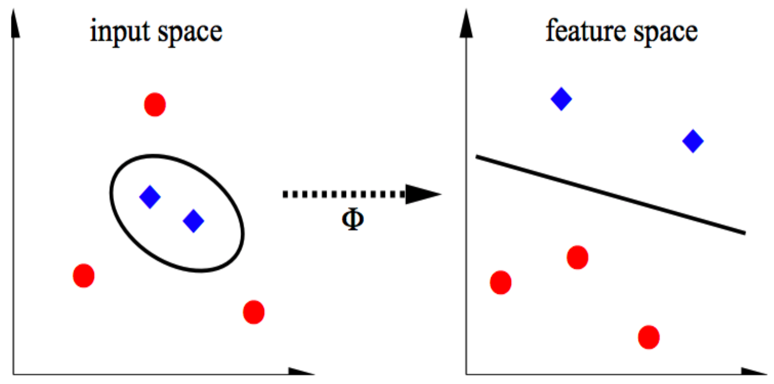


FIGURE 5.7: The circles and diamonds are not linearly separable in the input space ; they can be linearly separable in feature space.

Therefore, the non-linear model is built by two steps: firstly, a non-linear mapping projects the data into a feature space $\mathbb{F}$, and secondly a linear model is used to do classification in the feature space. So the decision function from Eq.(5.33) is transformed into:

$$f_d(\vec{F}_{\text{new}}) = \sum_{i=1}^{k} \vec{\beta}^* y_i \langle \Phi(\vec{F}_i)^T, \Phi(\vec{F}_{\text{new}}) \rangle + b^* \tag{5.34}$$

where $\Phi : \vec{F} \rightarrow \mathbb{F}$ is a non-linear map from the input space to feature space and $\langle ., . \rangle$ denotes the inner product. Note that both Eq.(5.34) and Eq.(5.34) only depend on the mapped data through dot products in the feature space $\mathbb{F}$. Let K(. , .) be defined the kernel function as:

$$K(\vec{F}_i, \vec{F}_{\text{new}}) = \langle \Phi(\vec{F}_i)^T, \Phi(\vec{F}_{\text{new}}) \rangle \tag{5.35}$$

The mapping function Φ become unnecessary when a kernel function is defined which directly calculates the value of the dot product of the mapped data points in the feature space $\mathbb{F}$. The advantage of such a kernel function is that the complexity of the optimization problem remains only dependent on the dimensionality of the input space and not of the feature space. Then the decision function is rewritten to:

$$f_d(\vec{F}_{\text{new}}) = \sum_{i=1}^{k} \vec{\beta}^* y_i K(\vec{F}_i, \vec{F}_{\text{new}}) + b^* \tag{5.36}$$

A kernel function can be interpreted as a kind of similarity measure between the input objects In practice, the following kernel functions in Table 5.3 are used for most of the applications.

Kernel Type	Kernel Function
Linear	$K(F^i, F^j) = F^i F^j$
Polynomial	$K(F^i, F^j) = (aF^i F^j + \theta)^d$
Radial Basis Function (RBF)	$K(F^i, F^j) = e^{-\frac{\lvert F^i - F^j \rvert^2}{2\sigma^2}}$
Sigmoid	$K(F^i, F^j) = tanh(aF^i F^j + \theta)$

TABLE 5.1: Summary of kernel functions

5.3.4 Vehicle Types Classification By CNN

CNN are primarily used for 2D image recognition, so the architecture of CNN is illustrated on a 2D image consisting of pixels which contain the color information represented by multiple channels such as RGB channels. In our application, the 3D point cloud data is projected into 2D planes of size 128×128 as shown in Figure 5.8. Each pixel value is ether 0 or 1 which represents whether it contains point or not. For the sake of simplicity, only the side view (y-z projection) is considered while explaining the model. Our approach to vehicle types classification involves processing several layers of CNN. This section will discuss the network architecture of CNN.

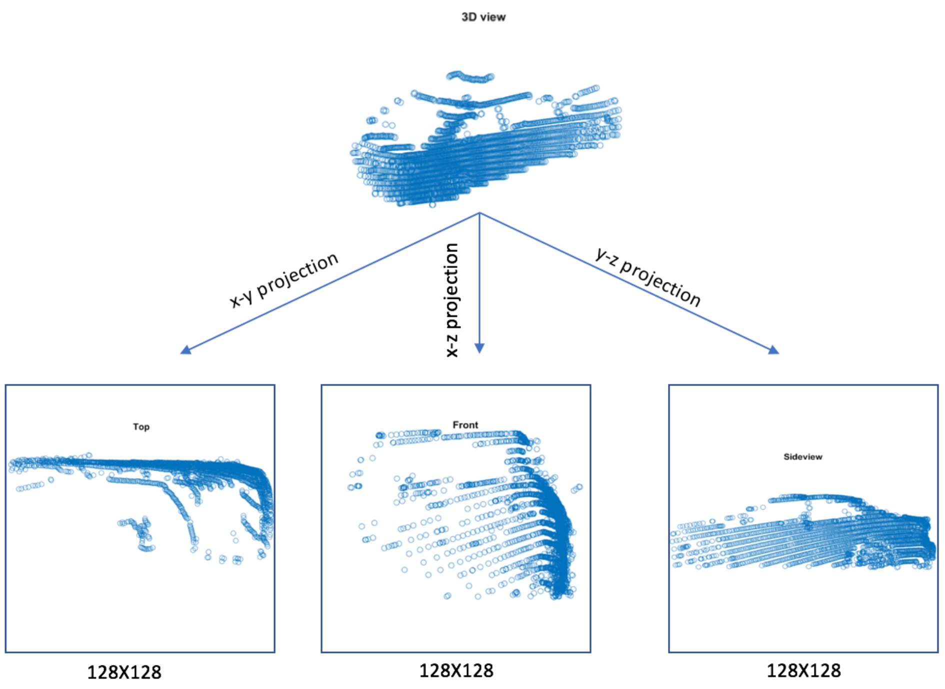


FIGURE 5.8: 3D view is projected into x-y, x-z, yz planes

A CNN is typically comprised of three main types of layers: (i) the convolution layers, (ii) the max pooling layers, and (iii) the fully connected layers. The layers of CNN are arranged in a feed forward structure: each convolution layer is followed by a max pooling layer, and the last fully connected layer is followed by the output as shown is Figure 5.9. The output consists of 4 different results: Sedan, SUV, Van, and Truck. The convolution and max pooling layers are 2D layers, whereas the fully connected layer is considered as a 1D layer. In CNN, each layers is made up of several planes, a plane is a 2D array of neurons for the convolution and max pooling layers, and 1D array of neurons for the fully connected layers. The output of a plane is called a feature map.

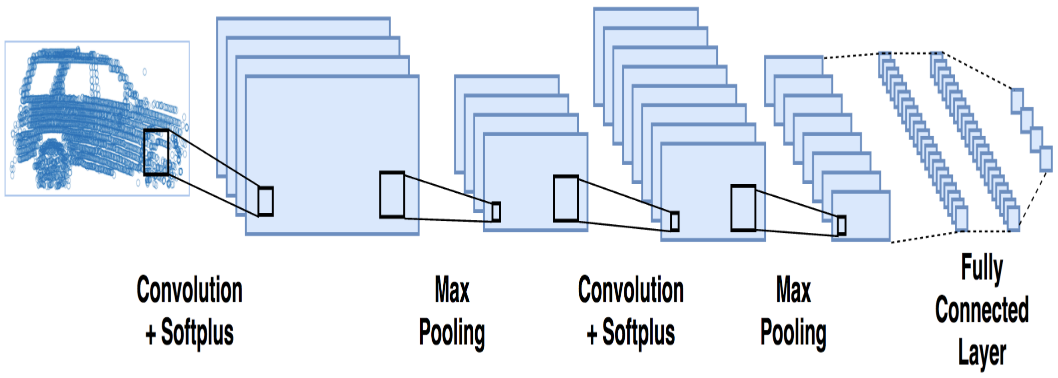


Figure 5.9: The data flow of CNN

1. **The Convolutional Layer**: In a convolutional layer, each plane is connected to one or more feature maps of the preceding layer. Each plane first computes the convolution between its 2D inputs and its adjustable weights. Thus the convolution outputs are summed together and added to an adjustable bias term. an activation function is applied to the result of convolution output to obtain the output feature map as shown in Figure 5.10. A convolution layer produces one or more feature maps. Each feature map is then connected to one plane in the next max pooling layer.

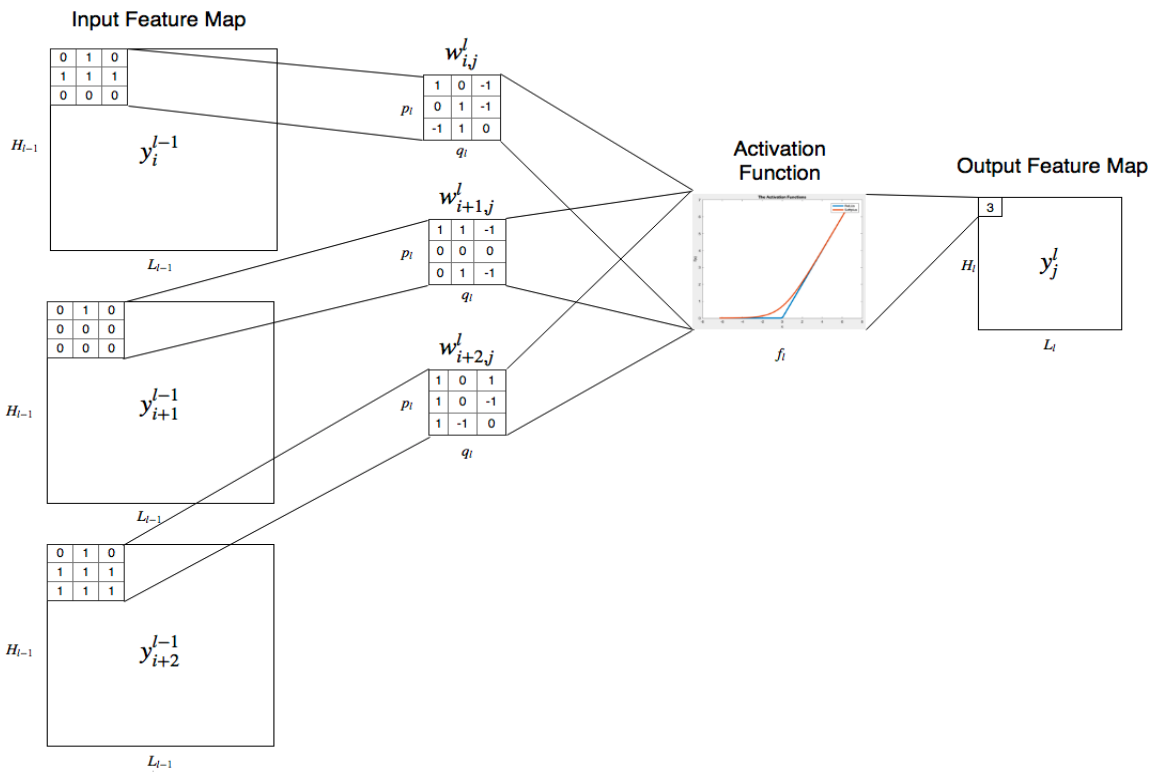


FIGURE 5.10: The example of convolution layer

Suppose the size of input feature map y_i^{l-1} is $H_{l-1} \times W_{l-1}$ at layer l, and $w_{i,j}^l$ be denoted as the convolution mask from feature map y_i^{l-1} in layer $l-1$ to feature map y_j^l in layer l. The size of weight $w_{i,j}^l$ is $p_l \times q_l$. Thus the feature map y_j^l in layer l is calculated as:

$$y_j^l = f_l(b_j^l + \sum_j w_{i,j}^l \otimes y_i^{l-1}) \tag{5.37}$$

where b_j^l is the adjustable bias term, and $\otimes$ denotes the convolution operator. The activation function f_l is usually implemented as rectified linear units (ReLU) which is shown to operate more robustly and efficiently [74].

However, the ReLu function is not differentiable at the origin, which makes it hard to use with back propagation training. Instead, a smooth version called the Softplus function is used in practice:

$$f(x) = \ln(1 + e^x) \tag{5.38}$$

where the derivative of the softplus function is the sigmoid function, as below:

$$f'(x) = \frac{d(\ln(1 + e^x))}{dx} = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} \tag{5.39}$$

Because zero padding for the input feature map is not performed in our application, so the size of output feature mapy_j^l is $(H_{l-1} - p_l + 1) \times (W_{l-1} - q_l + 1)$.

2. **The Max Pooling Layer**: The max pooling layer is used after the convolution layer. The operation performed by this layer is also called "downsampling", as the reduction of spatial dimension leads to loss of information as well. However, such a loss is beneficial to the network for two reasons:

   (a) the decrease in size leads to less computational time for the upcoming layers of the network.

   (b) it work against over-fitting.

   the max pooling layer takes a sliding window or a certain region that is moved across the input feature map by taking the maximum value in the window as shown in Figure 5.11. The size of sliding window in our application is 2×2. Therefore, for a input feature map y_j^l which has a size of $H_l \times W_l$, the size of output feature map y_j^{l+1} is $H_{l+1} = \frac{H_l}{2}$, and $W_{l+1} = \frac{W_l}{2}$.

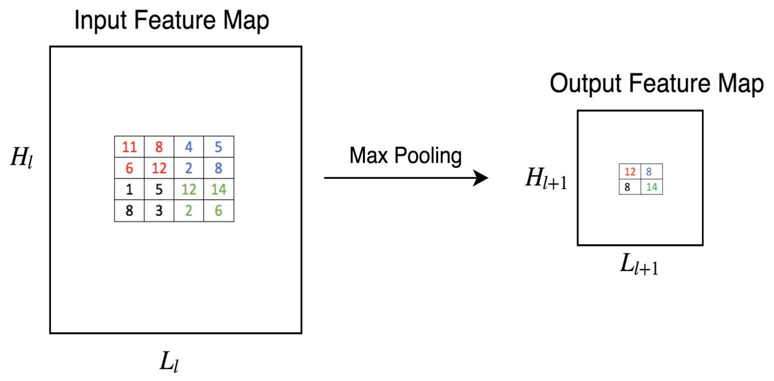


FIGURE 5.11: The example of max pooling layer

3. **The Fully Connected Layer**: The fully connected layer is configured such that every feature map from the last max pooling layer is connected to every layer of the fully-connected layer as shown in Figure 5.12. The fully connected layers are typically used in the last stage of the CNN to construct the desired number of outputs.

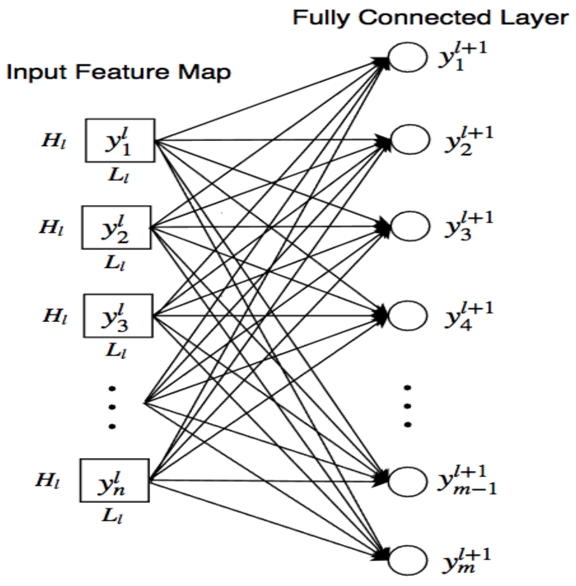


FIGURE 5.12: The example of fully connected layer

Let $w_{n,m}^{l+1}$ denote the weight from feature map n of the last max pooling layer to the neuron m of the fully connected layer $l+1$, and let b_m^{l+1} be the bias term. The output y_m^{l+1} of the neuron m of the fully connected layer is computed as:

$$y_m^{l+1} = f_l(\sum_1^n w_{n,m}^{l+1} \cdot y_n^l + b_m^{l+1}) \tag{5.40}$$

The output layer L follows the result of the fully connected layers. In our application, the output layer consists of 4 neurons. Therefore, the neurons of the output layer is calculated as:

$$\begin{aligned}
y_1^L &= f_l(\sum_1^n w_{n,1}^L \cdot y_n^{L-1} + b_1^L) \\
y_2^L &= f_l(\sum_1^n w_{n,2}^L \cdot y_n^{L-1} + b_2^L) \\
y_3^L &= f_l(\sum_1^n w_{n,3}^L \cdot y_n^{L-1} + b_3^L) \\
y_4^L &= f_l(\sum_1^n w_{n,4}^L \cdot y_n^{L-1} + b_4^L)
\end{aligned} \tag{5.41}$$

5.4 Experiments and Analysis

5.4.1 Data Preparation

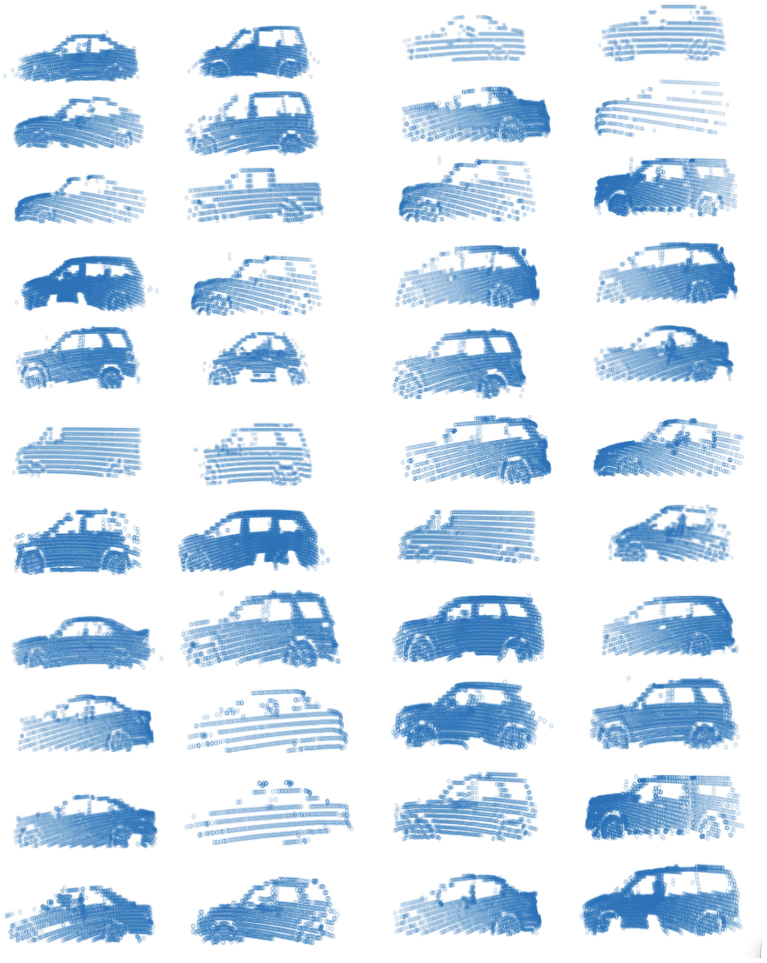


FIGURE 5.13: The data set for classification

To validate the proposed method, there are 2,660,000 points and 5320 clusters are collected manually to prepare for the ground-truth of vehicle types as shown in Figure 5.13. The data were collected by our real time system which is described in Chapter 2. The data sets used in our experiments are summarized in Table 5.2.

	Training Set	Validation Set	Testing Set
Sedan	40	100	2332
SUV	40	100	1375
Van	40	100	1047
Truck	40	100	316

TABLE 5.2: The vehicle data set

5.4.2 Parameter Selection for SVM

The C-Support Vector Classification (C-SVC) is used to allow imperfect separation of classes with penalty multiplier C for outliers [60]. The performance of the C-SVC depends on two parameters: c is a penalty parameter for weighting classification errors and g is a kernel function parameter. As it is not known beforehand which c and g are best for a given problem. One of the versions of this procedure is known as K-fold cross validation as shown in Figure 5.14.

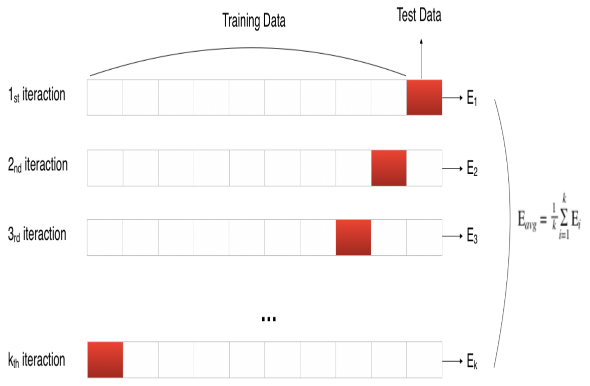


FIGURE 5.14: K-fold cross validation

In K-fold cross validation, the training set is divided into k subsets of equal size, and one subset is tested trained on the remaining $k-1$ subsets. For each grid cell, one pairs of (c, g) values are tried and evaluated and the one with the best K-fold cross validation accuracy is picked. A practical method to choose c and g values is trying exponentially growing sequences (for example, $c = 2^{-5}, 2^{-3}, ..., 2^{5}$, $g = 2^{-15}, 2^{-13}, ..., 2^{3}$). The grid-search procedure is compared at shown in Table 5.3:

	Linear	Polynomial	RBF	Sigmoid
c	0.0625	2	4	0.0625
g	1	1	0.000976	0.000976
Time (s)	0.01	0.1	0.015	0.37
Accuracy	90.75%	91.2%	94.39%	91.96%

TABLE 5.3: Grid-search result comparison

From the Table 5.3, the RBF has obtained the best accuracy result while still maintaining fast processing time. Therefore, Radial Basis Function(RBF) is chosen as our kernel function.

5.4.3 Parameter Selection for CNN

Different network structures are investigated for the CNN classifier.

- Input size: 32x32, 64x64, 128x128.
- Activation function: Linear, ReLu, Softplus.

- Number of feature maps in each layers: 32, 64, 128.
- Learning Rate: 0.01, 0.05, 0.1, 0.7.

For each parameters listed above, the CNN network was constructed. In total, 108 networks were created and trained. After training, the 108 networks were evaluated on the validation set, and the best network was selected as shown in Table 5.4.

Learning Rate	0.05		
Layer	Input Size	Kernel Size	Output Size
Convolution	$128 \times 128 \times 1$	1×1	$128 \times 128 \times 64$
Max Pool	$128 \times 128 \times 64$	2×2	$64 \times 64 \times 64$
Convolution	$64 \times 64 \times 64$	1×1	$64 \times 64 \times 128$
Max Pool	$64 \times 64 \times 128$	2×2	$32 \times 32 \times 128$
Fully	$32 \times 32 \times 128$	-	$1 \times 1 \times 2048$
Fully	$1 \times 1 \times 2048$	-	$1 \times 1 \times 2048$
Output	$1 \times 1 \times 2048$	-	1×4

Table 5.4: The CNN parameter selection

5.4.4 Performance Evaluation

This classification produces four outcomes – true positive, true negative, false positive and false negative as shown in Figure 5.15:

- True positive (TP): correct positive prediction
- False positive (FP): incorrect positive prediction
- True negative (TN): correct negative prediction
- False negative (FN): incorrect negative prediction

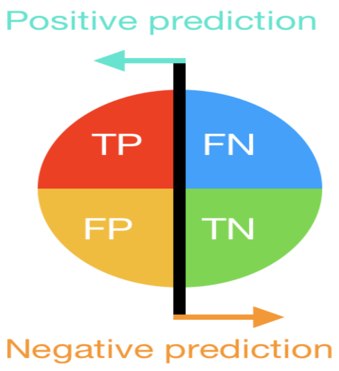


FIGURE 5.15: Classification produces four outcomes – true positive, false positive, true negative, and false negative.

The quantitative performance of our PCA-SVM-CNN model are analyzed using the metrics in Equation 5.42 to 5.47 [75].

1. **Sensitivity**: Sensitivity (SN) is calculated as the number of correct positive predictions divided by the total number of positives:

$$\text{SN} = \frac{TP}{TP + FN} \tag{5.42}$$

2. **Specificity**: Specificity (SP) is calculated as the number of correct negative predictions divided by the total number of negatives:

$$\text{SP} = \frac{TN}{FP + TN} \tag{5.43}$$

3. **Precision**: Precision (PREC) is calculated as the number of correct positive predictions divided by the total number of positive predictions:

$$\text{PREC} = \frac{TP}{TP + FP} \tag{5.44}$$

4. **False Positive Rate** : False Positive Rate (FPR) is calculated as the number of incorrect negative predictions divided by the total number of negatives:

$$\text{FPR} = \frac{FP}{FP + TN} \tag{5.45}$$

5. **F-score** : F-score is a harmonic mean of precision and recall:

$$\text{F-score } = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{5.46}$$

6. **Error rate**: Error rate (ERR) is calculated as the number of all incorrect predictions divided by the total number of the dataset:

$$\text{ERR} = \frac{FP + FN}{TP + TN + FP + FN} \tag{5.47}$$

In classification analytics, a table of confusion is a table that reports the number of false positives, false negatives, true positives, and true negatives. This allows more detailed analysis than the general accuracy. So the confusion matrix and the performance results are shown in Table 5.5 and Table 5.6, respectively:

	Predicted				
		Sedan	SUV	Van	Truck
Actual	Sedan	**2268**	44	13	7
	SUV	30	**1004**	328	13
	Van	15	41	**980**	11
	Truck	14	25	24	**253**

TABLE 5.5: The confusion matrix

Metrics	Sedan	SUV	Van	Truck
TP	2268	1004	980	253
TN	2237	3501	3525	4252
FP	59	110	365	31
FN	64	371	67	63
SN	97.25%	75.02%	93.60%	80.06%
SP	97.43%	96.95%	90.62%	99.28%
PREC	97.46%	90.13%	72.86%	89.08%
FPR	2.57%	3.05%	9.38%	0.72%
F-score	97.36%	80.67%	81.94%	84.33%
ERR	2.66%	9.86%	9.75%	2.04%
Overall Accuracy		88.87%		

TABLE 5.6: The analysis of classification result

From the prediction result of confusion matrix, the most challenged pair is the of class “SUV” and class “Van”. This is because they have quite similar appearances in terms of shape and size as shown in Figure 5.16. From Table 5.6, the best result of vehicle types classification is the class of "Truck" and "Sedan". This is because the shape and size of these two types are quite different from others vehicle types.

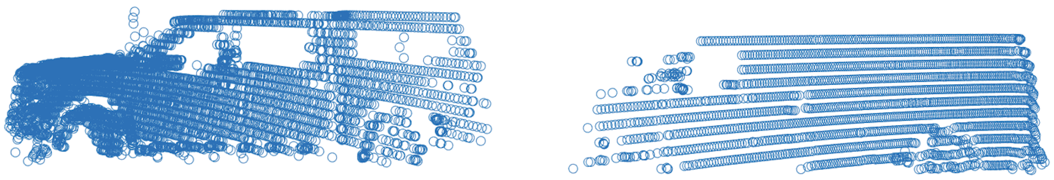

(A) The "SUV" class (B) The "Van" Class

FIGURE 5.16: The most two confusing classes

The conventional classification methods, namely an end-to-end CNN, end-to-end SVM is used to compared with our proposed method as shown in Table 5.7.

- End-to-end CNN: The original data are trained by CNN directly to predict the vehicle types.
- End-to-end SVM: The original data are trained by CNN directly to predict the vehicle types.
- PCA-SVM-CNN: The principle features are selected by CNN from the original data and classified as either "Vehicle" or "Non-vehicle". Then the "Vehicle" data set are trained by CNN to

Approach	Overall Accuracy
End-to-end CNN	83.23%
End-to-end SVM	74.54%
PCA-SVM-CNN	88.87%

TABLE 5.7: The comparison of classification result

From Table 5.7, the results demonstrate the improved performance of the proposed PCA-SVM-CNN classification approach for LiDAR based system. The reasons that integrating PCA and SVM lead to better performance are as follows.

1. Although the real time experiment is performed on highway scenarios where the surrounding clusters are most likely "Vehicle" clusters, the cluster sets often contain some "Non-vehicle" objects. These "Non-vehicle" cluster might be bushes along the road, insufficient number of points from distance or incomplete clusters because of partially block by objects. Because the "Non-vehicle" clusters vary in shape, size, density, intensity and so on. It is difficult to define the "Non-vehicle" class to be trained by CNN. As a result, the performance of the end-to-end CNN or end-to-end SVM is degraded when the collected data set contains "Non-vehicle" clusters in a dynamics environment.

2. To better accomplish the classification task, the feature extraction methods such as PCA is used to extract the significant features of vehicles and as a preprocessing tool in the the binary classification using SVM. As a result, the "Non-vehicle" clusters are eliminated which will certainly improve performance accuracies for vehicle types classification on the "Vehicle" data sets.

5.5 Conclusion

In this chapter, a supervised learning approach based on our LiDAR sensor for vehicle type classification is proposed. This approach is referred to the PCA-SVM-CNN. The proposed approach was implemented on our real time LiDAR system. The method takes the results from previous chapter as the input and outputs the vehicle types. The different types of vehicle determine the accurate center location of the vehicle which is used for the position estimation.

Chapter 6

The Algorithm: Design of Tracking System

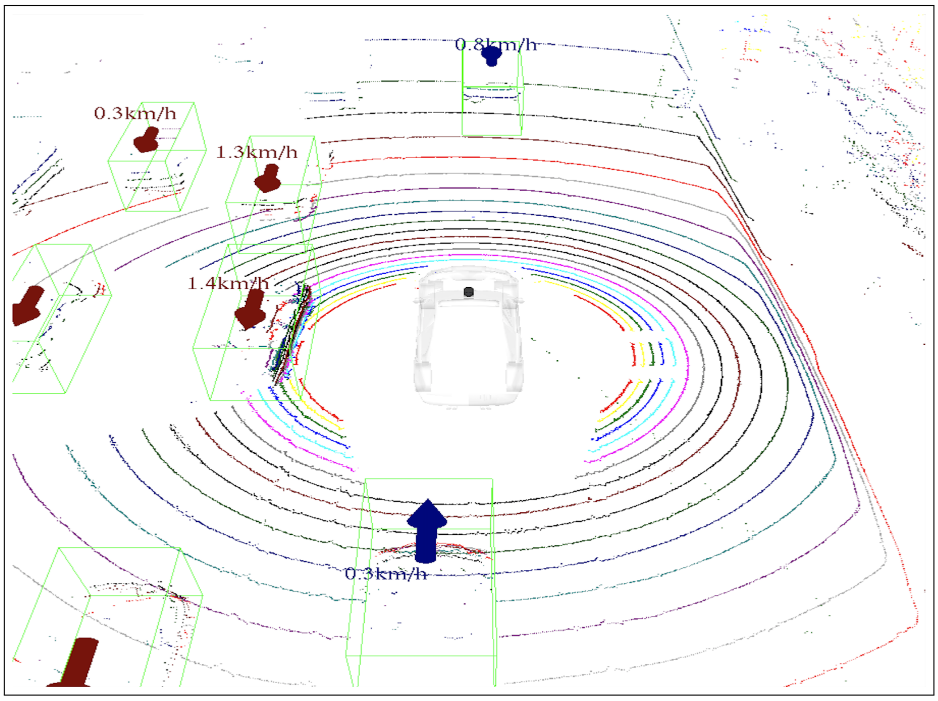


FIGURE 6.1: The result of tracking system

6.1 Introduction

The measurements from sensors provide information on the surrounding environment, also contains clutter data and noise. To handle disturbances and motion uncertainties, the Smooth Variable Structure Filter (SVSF) can be used as introduced in 2007. The SVSF is designed based on the sliding mode concept and manages to maintain the estimated states within a subspace of true states. A generalized version of SVSF (GSVSF) is developed to handle vehicle tracking when there are fewer number of measurements than states in real time scenarios. The LiDAR 3D data consists of both dense and sparse data in each scan that makes the data processing and object tracking more challenging. Therefore, a generalized SVSF based method is proposed that is a combination of the Hungarian algorithm (HA) and Probability Data Association Filter (PDAF) referred to the GSVSF-HA/PDAF. The proposed hybrid method is applied to our LiDAR real time system. In addition, a comparative analysis is implemented to compare the proposed method with Kalman filter (KF) based methods. Figure 6.1 shows an example of the results of our proposed tracking system.

In order to monitor the surrounding of vehicles and to proactively react to potential traffic infringements, it is necessary to track the motion states of surrounding vehicles. Given the geometry of the LiDAR that is mounted on the roof of a vehicle as described in Chapter 2, the frame is processed through a ring-like pattern as shown in Figure 6.2. However, each scan of LiDAR data includes both dense data and sparse data as shown in Figure 6.3, is a challenge for LiDAR sensor to decide results of originated in noise. Noise in turn degrade the capability that continuously and reliably detecting objects, and recursively estimating the state of the surrounding vehicles. To address this problem, a generalized SVSF based hybrid approach is presented for real time traffic tracking and estimation with our LiDAR System.

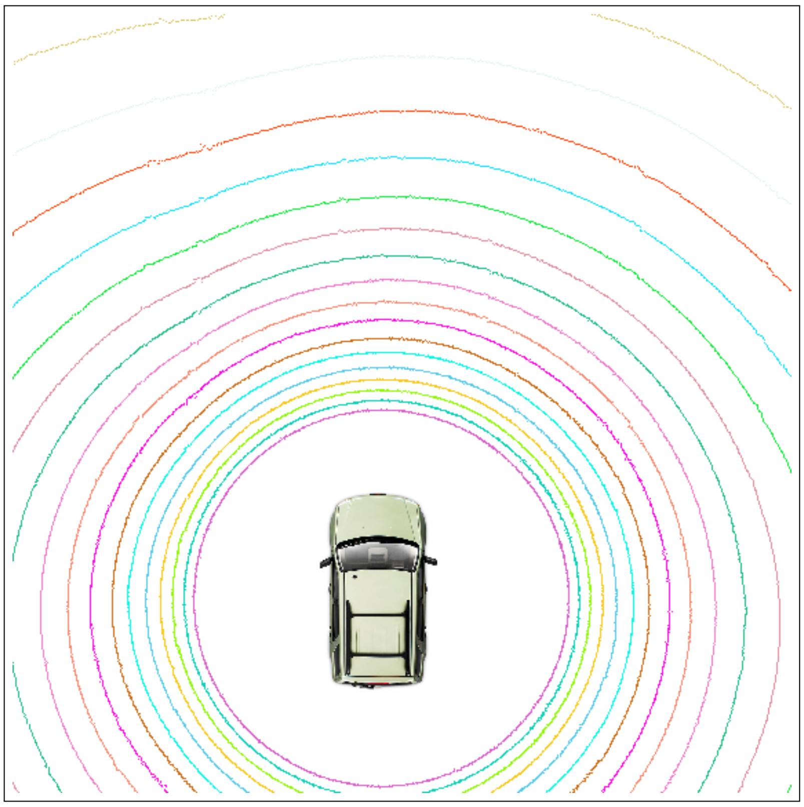


Figure 6.2: The ring-like pattern

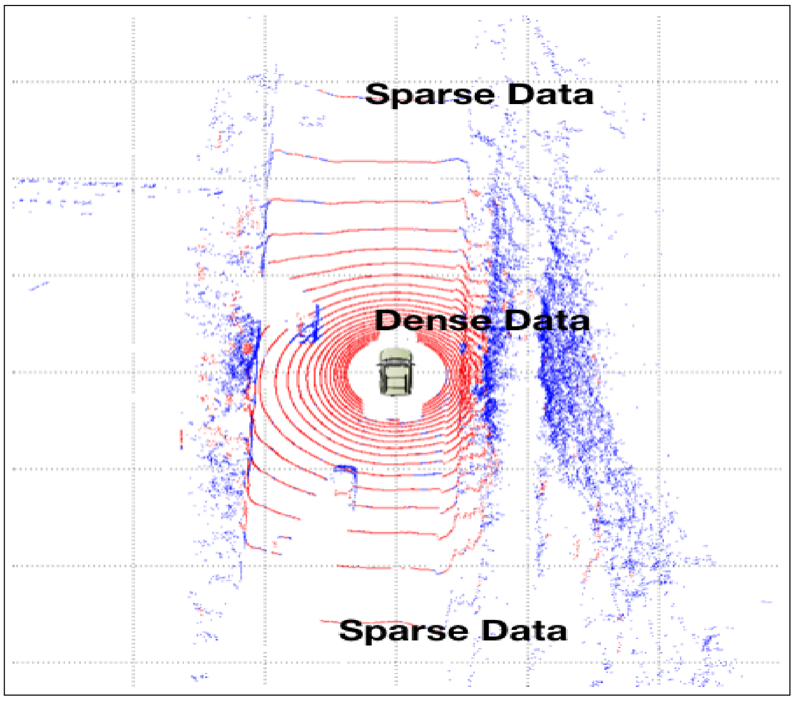


FIGURE 6.3: The process of tracking system

Table 6.1 lists the parameters and their associated definition that are used in the following sections.

Parameters	Definition
x	State vector
z	Measurement (system output) vector
w	System noise vector
v	Measurement noise vector
A	Linear system transition matrix
B	Linear input gain matrix
H	Linear measurement (output) matrix
K	Filter gain matrix
P	State error covariance matrix
Q	System noise covariance matrix
R	Measurement noise covariance matrix
$\hat{e}$	Measurement (output) error vector
γ	SVSF memory term
ϕ	SVSF smoothing boundary layer
diag[a]	Diagonal of some vector or matrix a
sat()	Saturation function
|a|	Absolute value of a
a^T	Transpose of the vector a
$\circ$	Denotes element-by-element multiplication
$k\|k-1$	The a priori state
$k\|k$ or $k-1\|k-1$	The a posteriori state

Table 6.1: List of Important Nomenclature and Parameters

6.2 Related Work

In real time application, measurements are often contaminated by clutter and noise. As such the tracking strategy needs to be robust to remove noise and uncertainties. The well-known Kalman filter (KF) was introduced in the 1960s and yields a statistically optimal solution for linear estimation problems in the presence of Gaussian noise [76]. It is formulated in a predictor-corrector manner, and is implemented recursively. The optimality of the KF comes at the price of stability and robustness [77]. For vehicle tracking, different approaches have been proposed in the literature. Many of them rely on the implementation of an Extended Kalman filter (EKF) [78] [79] [80]. The performance of the EKF is reliable in many practical situations, but the non-linear state equations may lead to instability problems when dealing with the problem of maneuvering targets in a densely cluttered environment. To address the modeling uncertainty problem, the Smooth Variable Structure Filter (SVSF) was introduced in 2007 [81]. It is a recursive predictor-corrector filter based on the sliding mode concept [82]. The significant features of SVSF are its robustness, multiple indicators of performance, and its ability to identify the source of uncertainty. Attari *et al.* [83] presented a generalized variable boundary layer (GVBL-SVSF), which has the optimal characteristics of the KF, while maintaining the robustness against modeling uncertainties inherited from SVSF.

The essence of multiple targets association is to deal with uncertainty that occurs when the sensor provides noisy measurements that is clutter-originated. In other word, it should be decided which measurement is associated to which target and which measurements are noise. In [84], detection and tracking were handled in a coupled manner by use of the Hungarian algorithm (HA). The HA is for assigning targets by a one-for-one matching to identify the lowest-cost solution. The one-for-one matching algorithms such as HA have been shown that they works reasonably well in dealing with measurement origin uncertainty of sparse scenarios from previous studies [85]. The Probabilistic Data Association Filter(PDAF) and the Joint Probabilistic Data Association Filter(JPDAF) were proposed to handle the measurement to target association problem in a probabilistic manner and consider all of the measurements that are falling into the validation gate [86]. Although JPDAF has less bias than the PDAF in a clean environment, it has been shown to have more coalescence and rejection bias phenomenon than PDAF in a cluttered environment [87]. However, the JPDAF method has an exponential increase in computational time with an increase in the number of tracked targets that makes it a less practical in real time applications [87].

Due to the assumption of PDAF/JPDAF that at most one measurement is originated from each target, gating failures arising when the new measurements does not lie within the gate cased by occlusions or frames lost. As a result, the PDAF/JPDAF tends to reject this measurement and generate a new track to handle it which leads to reduced accuracy and increase computation time. Figure 6.4 shows an example of gating failure that the measurement z_k does not lie within the gate of the track x_k. Therefore, a new track is assigned to handle the measurement z_k. In order to robustly track multiple targets in the real time environment where occlusions or frames lost are usually occurs, and to implement an effective real time LiDAR based tracking system, a method based on a combination of HA, PDAF, and SVSF is proposed.

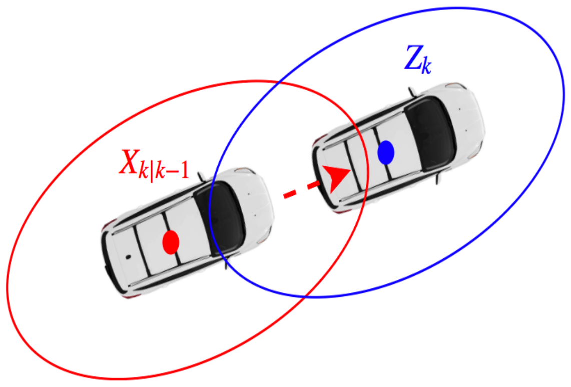


FIGURE 6.4: Example of measurement rejection due to gating failure from the same target

6.3 Tracking Strategy

6.3.1 The Kalman Filter Filter

The Kalman Filter is essentially a set of mathematical equations implement under a predictor-corrector cycle as shown in Figure 6.5, that is optimal in the sense that it minimizes the trace of the estimated state as covariance matrix when some presumed conditions are met.

The general problem is to estimate the state vector $x \in \Re^n$ of a discrete time dynamic system. Such a system is defined by using a set of linear stochastic

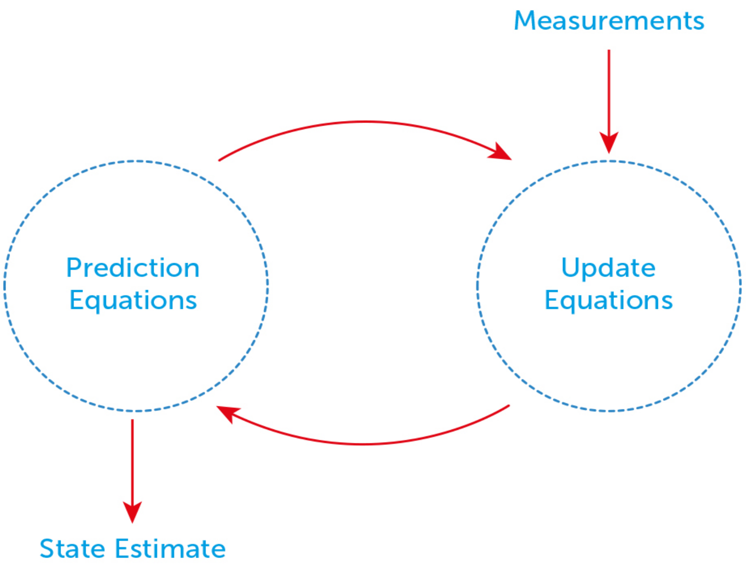


FIGURE 6.5: Predictor-Corrector cycle

difference equations as follows:

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_k + w_k \\ z_k &= Hx_k + v_k \end{aligned} \tag{6.1}$$

- $x_k \in \Re^n$ is the state
- $u_k \in \Re^l$ is the input vector
- $w_k \in \Re^p$ is the measurement noise
- $z_k \in \Re^m$ is the measured output
- $v_k \in \Re^p$ is the process noise or disturbance

Note that the $n \times n$ matrix A relates its system matrix, the $m \times n$ matrix H relates its output matrix, and the $n \times l$ matrix B is the input matrix. The w_k and v_k represent the process and measurement noise which assumed to be white noise zero mean and with normal probability distributions such that:

$$\begin{aligned} p(w) &\sim N(0, Q) \\ p(v) &\sim N(0, R) \end{aligned} \tag{6.2}$$

Define $\hat{x}_{k|k-1} \in \Re^n$ as the a priori state estimate at step k given information at step $k-1$. Also define $\hat{x}_{k|k} \in \Re^n$ as a posteriori state estimate at time k given measurement z_k. Thus the a priori and the a posteriori state estimation errors are defined as:

$$\begin{aligned} \hat{e}_{k|k-1} &= x_k - \hat{x}_{k|k-1} \\ \hat{e}_{k|k} &= x_k - \hat{x}_{k|k} \end{aligned} \tag{6.3}$$

The a priori estimation error covariance and the a posteriori estimation error covariance are then defined as:

$$\begin{aligned} \hat{P}_{k|k-1} &= E[\hat{e}_{k|k-1}\hat{e}^T_{k|k-1}] \\ \hat{P}_{k|k} &= E[\hat{e}_{k|k}\hat{e}^T_{k|k}] \end{aligned} \tag{6.4}$$

The a posteriori state estimate $\hat{x}_{k|k}$ is computed as a linear combination of the a priori estimated state $\hat{x}_{k|k-1}$ and a weighted difference between the actual measurement z_k and the predicted measurement $H\hat{x}_{k|k-1}$, as in Eq.(6.5).

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H_k\hat{x}_{k|k-1}) \tag{6.5}$$

where $\hat{e}^I_{k|k-1} = z_k - H\hat{x}_{k|k-1}$ is the measurement innovation term which reflects the discrepancy between the predicted measurement and the actual measurement. So when the innovation term is equal to zero, it means that these two are in complete agreement. K_k is defined as the Kalman gain at current step k that minimizes the a posteriori estimation error covariance in Eq.(6.4). The optimal gain K_k can be obtained as [88]:

$$\begin{aligned} K_k &= \hat{P}_{k|k-1}H^T(H\hat{P}_{k|k-1}H^T + R)^{-1} \\ &= \frac{\hat{P}_{k|k-1}H^T}{H_k\hat{P}_{k|k-1}H^T + R} \end{aligned} \tag{6.6}$$

From Eq.(6.6), it can be observed that if the measurement noise covariance, R approached zero, then $\lim\limits_{R\to 0} K_k = \frac{1}{H}$, and in the KF, then the actual measurement is trusted more than the predicted state using a model. On the other hand, as the a priori estimation error covariance $\hat{P}_{k|k-1}$ approaches zero, then $\lim\limits_{\hat{P}_{k|k-1}\to 0} K_k = 0$, and the actual measurement is trusted less than the predicted state using a model.

The process of KF can be divided into 2 steps: (i) time update, and (ii) measurement update. The time update involves using a system model to project its estimate states one step forward to obtain the a priori estimates. So this step can be thought of as the predictor. The second part, the a priori states are used to predict the output, and this preidction is then compared to the actual measurements to obtain the a posteriori estimates. So this part can be thought of as the corrector as shown in Figure 6.6.

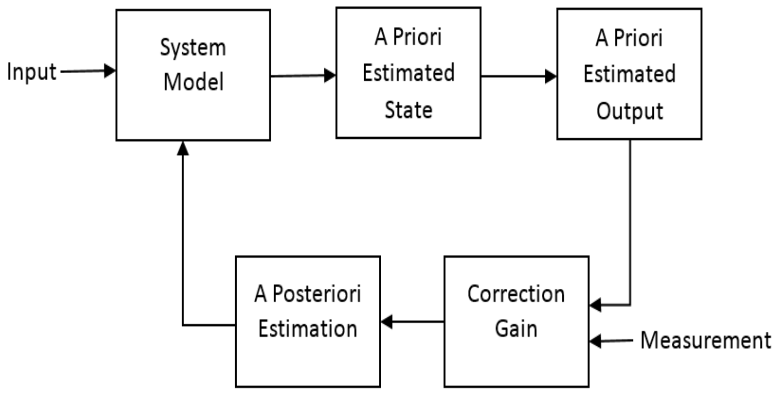


FIGURE 6.6: The process flow of Kalman filter

The KF equations are presented below where $\mathbb{I}$ represents the identity matrix:

Time Update

$$\begin{aligned} \hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} + Bu_k \\ \hat{P}_{k|k-1} &= A\hat{P}_{k-1|k-1}A^T + Q \end{aligned} \tag{6.7}$$

Measurement Update

$$\begin{aligned} K_k^{kf} &= \hat{P}_{k|k-1} H^T (H \hat{P}_{k|k-1} H^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k^{kf} (z_k - H \hat{x}_{k|k-1}) \\ \hat{P}_{k|k} &= (\mathbb{I} - K_k^{kf} H) \hat{P}_{k|k-1} \end{aligned} \tag{6.8}$$

6.3.2 The Extended Kalman Filter

Unfortunately, system are rarely linear in practice. The Extended Kalman filter (EKF) relaxes the linearity assumption of KF. In the EKF the state transition and measurement probabilities are defined as nonlinear stochastic difference equations such that:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) \\ z_k &= h(x_k, v_k) \end{aligned} \tag{6.9}$$

where x_k is the state vector, z_k the measurement vector, $w \sim N(0, Q)$ and $v \sim N(0, Q)$ all at time step k.

The Taylor series expansion [78] is used to linearize the nonlinear function $f(x)$ around the most recent state estimation $\hat{x}$ as follows:

$$f(x) = f(\hat{x}) + \frac{\partial f}{\partial x}(x - \hat{x}) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x - \hat{x})^2 + \frac{1}{6}\frac{\partial^3 f}{\partial x^3}(x - \hat{x})^3 + \dots \tag{6.10}$$

This can be written as

$$f(x) = f(\hat{x}) + \frac{\partial f}{\partial x}(x - \hat{x}) + \text{higher order terms} \tag{6.11}$$

For x sufficiently close to the equilibrium point $\hat{x}$, the higher order terms are assumed to be negligible and then:

$$\begin{aligned} x_k &\approx f(\hat{x}_{k-1}) + A(x_{k-1} - \hat{x}_{k-1}) + w_k \\ z_k &\approx h(\hat{z}_k) + H(x_k - \hat{x}_k) + v_k \end{aligned} \tag{6.12}$$

where A is the Jacobian of function f with respect to x evaluated at $\hat{x}_{k-1}$ and C is the Jacobian of function h with respect to x evaluated at $\hat{x}_k$

$$\begin{aligned} A &= \frac{\partial f}{\partial x}(\hat{x}_{k-1}, u_k, 0) \\ C &= \frac{\partial h}{\partial x}(\hat{x}_k, 0) \end{aligned} \tag{6.13}$$

Therefore, the EKF algorithm can be summarized as follows:

Time Update

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_k, 0) \\ \hat{P}_{k|k-1} &= A\hat{P}_{k-1|k-1}A^T + Q \end{aligned} \tag{6.14}$$

Measurement Update

$$\begin{aligned} K_k^{ekf} &= \hat{P}_{k|k-1}H^T(H_k\hat{P}_{k|k-1}H^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k^{ekf}(z_k - h(\hat{x}_{k|k-1}, 0)) \\ \hat{P}_{k|k} &= (I - K_k^{ekf}H)\hat{P}_{k|k-1} \end{aligned} \tag{6.15}$$

where A_k is the Jacobian matrix of partial derivatives f with respect to $\hat{x}_{k-1|k-1}$ and H_k is the Jacobian matrix of partial derivatives h with respect to $\hat{x}_{k|k-1}$.

6.3.3 The Smooth Variable Structure Filter

The smooth variable structure filter (SVSF), similar to the Kalman filter, has a predictor-corrector cycle, and is based on sliding mode concepts for state estimation[81]. The algorithm can guarantee stability given bounded uncertainties. This estimation method includes an inherent switching functions. The estimated states will converge towards the true state trajectory and then remain within an existence subspace around the trajectory. The basic estimation concept of the SVSF is shown in Figure 6.7.

Consider the following system where f can be either linear or nonlinear and H is a linearized output matrix.

$$\begin{aligned} x_k &= f(x_{k-1}, u_k) + w_k \\ z_k &= Hx_k + v_k \end{aligned} \tag{6.16}$$

Where x_k is the system's state, u_k is the input, w_k is the system noise, z_k is the measurement output and v_k is the measured noise. At each time step, the SVSF starts by calculating the a priori state estimates:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{6.17}$$

The corresponding predicted measurement $\hat{z}_{k|k-1}$ and the measurement error $e_{k|k-1}$ are computed as:

$$\begin{aligned} \hat{z}_{k|k-1} &= H\hat{x}_{k|k-1} \\ \hat{e}_{k|k-1} &= z_k - \hat{z}_{k|k-1} \end{aligned} \tag{6.18}$$

The next step is the calculation of the SVSF corrective gain. In Figure 6.7, the estimated states are forced towards the true state trajectory by a corrective gain K^{SVSF}. The estimated states would chatter and slide along the true trajectory while remaining within a neighborhood referred to as the existence subspace. The existence subspace width is a function of uncertainties and is time varying.

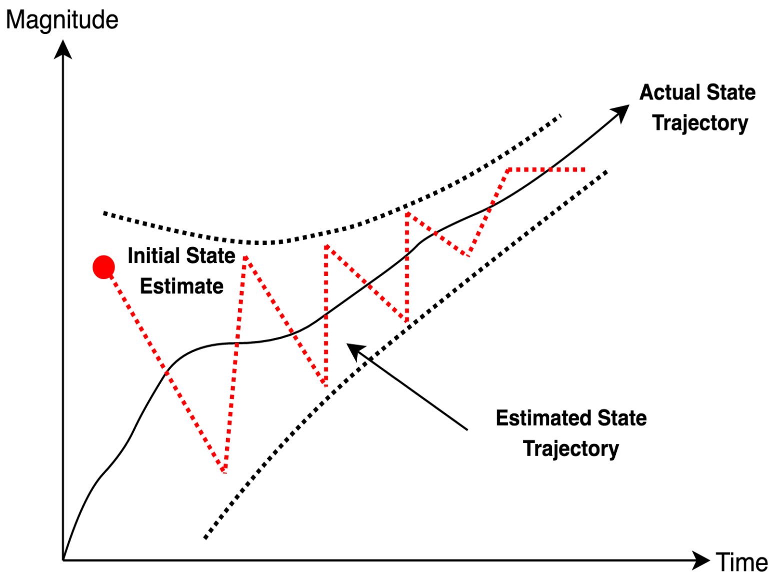


FIGURE 6.7: State estimation toward the true trajectory

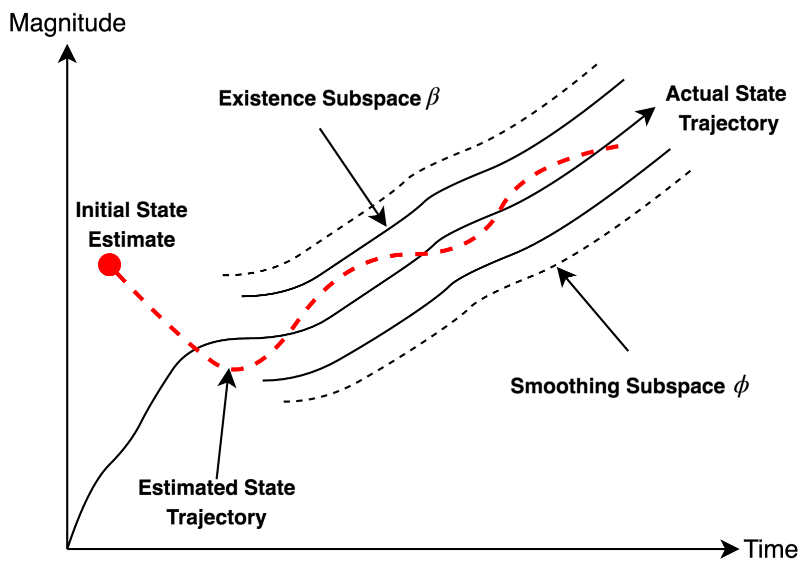
The SVSF's stability is proven by using a discrete *Lyapunov function* [81]. As a result, the estimation error decreases as the estimation process proceeds and both the estimated states and the estimated output will eventually converge. Using the *Lyapunov function*, the corrective gain is designed such that it makes sure that the estimation error keeps getting smaller by each iteration. Therefore, the estimation process is stable if

$$|\hat{e}_{k|k}| < |\hat{e}_{k-1|k-1}| \tag{6.19}$$

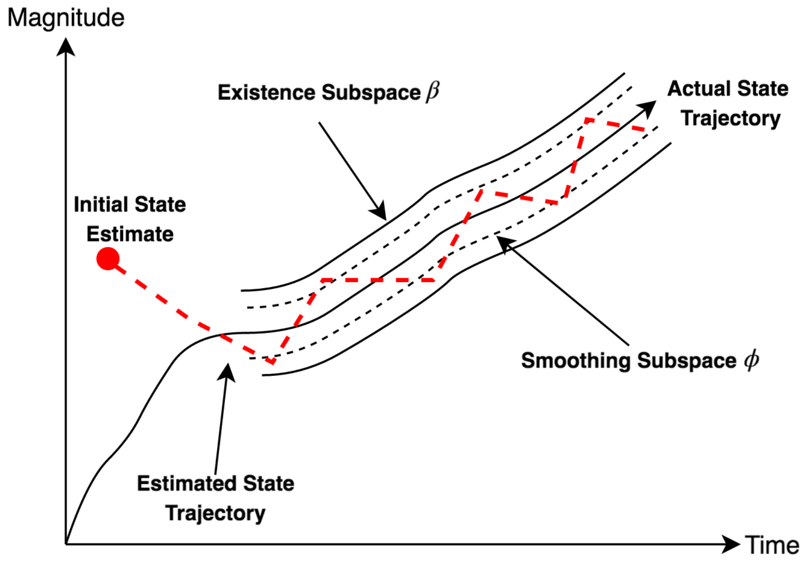
In order to satisfy this condition, the corrective gain is formulated as:

$$K_k^{SVSF} = H^{-1}(|\hat{e}_{k|k-1}| + \gamma|\hat{e}_{k-1|k-1}|)\text{sign}(\hat{e}_{k|k-1}) \tag{6.20}$$

where γ is the memory term, with a value between 0 and 1. The sign function generates chattering. However, the chattering can be removed by introducing a smoothing boundary layer ϕ_i. The selection of the width of the smoothing boundary layer reflects the level of uncertainties in the filter. The effects of the smoothing boundary layer are shown in Figure 6.8.


Magnitude
Existence Subspace β
Actual State Trajectory
Initial State Estimate
Smoothing Subspace ϕ
Estimated State Trajectory
Time

(A) Smooth estimated trajectory


Magnitude
Existence Subspace β
Actual State Trajectory
Initial State Estimate
Smoothing Subspace ϕ
Estimated State Trajectory
Time

(B) Chattering effect

FIGURE 6.8: The smooth boundary layer concept [81]

The width of the existence subspace β in Figure 6.8 reflects the amount of uncertainties in the estimation process in terms of modeling errors or noise. Typically, when the width of smoothing boundary layer is defined larger than the existence boundary $\phi > \beta$, the estimated state trajectory is smoothed. But when $\phi < \beta$, chattering effects remain because of the uncertainties being underestimated. Therefore, the sign function is replaced by a saturation function as follow:

$$\begin{aligned} K_k^{SVSF} &= H^{-1}(|\hat{e}_{k|k-1}| + \gamma|\hat{e}_{k-1|k-1}|) \circ \text{sat}(\frac{\hat{e}_{k|k-1}}{\phi}) \\ \text{sat}(\frac{\hat{e}_{k|k-1}}{\phi}) &= \begin{cases} \frac{\hat{e}_{k|k-1}}{\phi} & \text{for} |\hat{e}_{k|k-1}| \leq \phi \\ \text{sign}(\frac{\hat{e}_{k|k-1}}{\phi}) & \text{for} |\hat{e}_{k|k-1}| > \phi \end{cases} \end{aligned} \tag{6.21}$$

The performance of SVSF is significantly affected by choice of ϕ. If ϕ is chosen too large, the SVSF gain would approach zero. The a posteriori estimation therefore simply equals to the predicted state. As a conservative choice, ϕ can be chosen as the upper bound of the system uncertainty.

$$\phi = \tilde{A}_{max}(H_{max}^{-1}(z_{max} - v_{max})) + \tilde{B}_{max}u_{max} + w_{max} \tag{6.22}$$

The parameter $\tilde{A}_{max}, H_{max}$ and $\tilde{B}_{max}$ are the maximum error of the system and the measurement models. This is also called fixed smoothing boundary layer since the selected boundary layer width ϕ is larger than the existence subspace width β. Consequently, the estimated states are remaining within the boundary layer and the chattering effect is removed by applying the saturation function in Eq.(6.21).

Overall, the main steps of SVSF process are as following:

1. **Prediction of the states and measurements**

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_k) \\ \hat{z}_{k|k-1} &= H\hat{x}_{k|k-1} \end{aligned} \tag{6.23}$$

2. **Calculation of the measurement error**

$$\hat{e}_{k|k-1} = z_k - H\hat{x}_{k|k-1} \tag{6.24}$$

3. **Calculation of the corrective gain**

$$\begin{aligned} K_k^{svsf} &= H^{+}(|\hat{e}_{k|k-1}| + \gamma|\hat{e}_{k-1|k-1}|) \circ \text{sat}(\frac{\hat{e}_{k|k-1}}{\phi}) \\ \text{sat}(\frac{\hat{e}_{k|k-1}}{\phi}) &= \begin{cases} \frac{\hat{e}_{k|k-1}}{\phi} & \text{for}|\hat{e}^{I}_{k|k-1}| \leq \phi \\ \text{sign}(\frac{\hat{e}_{k|k-1}}{\phi}) & \text{for}|\hat{e}_{k|k-1}| > \phi \end{cases} \end{aligned} \tag{6.25}$$

4. **Update the state estimate**

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k^{svsf} \\ \hat{e}_{k|k} &= z_k - H\hat{x}_{k|k} \end{aligned} \tag{6.26}$$

6.3.4 Generalized SVSF

Utilizing a multiple model strategy with SVSF will increases the overall accuracy of the estimation process. A revised form of the SVSF was introduced in [89] such that the covariance matrix is included. Because it is unnecessary and expensive to have measurements associated with each of the states in the system. In [90] a generalized version of SVSF (GSVSF) is introduced to handle the cases when there are fewer number of measurements than states.

In practice, where the number of measurements are less than the order of the system, it is possible to obtain reduced order observers, refer as *Luenberger Observer* [91].

Assume that the state equation x_k can be partitioned and transformed into two parts: available and unavailable. Therefore, the state vector is transformed by using a transformation matrix, T such that:

$$Tx_k = \begin{bmatrix} x_k^u \\ x_k^l \end{bmatrix} = \begin{bmatrix} x_k^1 \\ \vdots \\ x_k^m \\ --- \\ x_k^{m+1} \\ \vdots \\ x_k^n \end{bmatrix} \tag{6.27}$$

where the upper segment x_k^u is the direct measurements and the lower segment x_k^l is the unmeasured states. Based on this division, the transformed state transition matrix is obtained as follows:

$$\Phi_k = TA_kT^{-1} = \begin{bmatrix} \Phi_k^{11} & \Phi_k^{12} \\ \Phi_k^{21} & \Phi_k^{22} \end{bmatrix} \tag{6.28}$$

The corresponding measurement matrix H_k is of dimension $m \times n$, where m is the number of measured states and n is the rank of the system such that $m < n$ is given by:

$$H_k = \begin{bmatrix} H_1 & H_2 \end{bmatrix} \tag{6.29}$$

where H_1 is of dimension $m \times m$ and H_2 is a null matrix of dimension $m \times (n - m)$. Then the state space equation should be revised for the estimation of states associated with H.

The a priori state error covariance matrix corresponding with the partitioned state vector is given by:

$$\hat{P}_{k|k-1} = \begin{bmatrix} \hat{P}^{11}_{k|k-1} & \hat{P}^{12}_{k|k-1} \\ \hat{P}^{21}_{k|k-1} & \hat{P}^{22}_{k|k-1} \end{bmatrix} \tag{6.30}$$

The corresponding partitioned of corrective gain is given as follow [81]:

$$K_k^{svsf} = \begin{bmatrix} K_k^u \\ K_k^l \end{bmatrix} \tag{6.31}$$

where K_k^u is defined as

$$\begin{aligned} K_k^u = H_1^{-1} \times \operatorname{diag}\Big[(|\hat{e}_{k|k-1}| + \gamma_u |\hat{e}_{k|k-1}|) \circ \operatorname{sat}(\frac{\operatorname{diag}(\hat{e}_{k|k-1})}{\phi})\Big] \\ \times \operatorname{diag}\Big[\hat{e}_{k|k-1}\Big]^{-1} \end{aligned} \tag{6.32}$$

where γ_u is an $m \times m$ diagonal matrix and the lower portion corrective gain K_k^l is defined as:

$$\begin{aligned} K_k^l = \operatorname{diag}\Big[|\Phi_k^{22}(\Phi_k^{12})^{-1}\hat{e}_{k|k-1}| + \gamma_l |(\Phi_k^{12})^{-1}\hat{e}_{k-1|k-1}|\Big] \\ \circ \operatorname{sat}(\frac{\Phi_k^{22}(\Phi_k^{12})^{-1}\operatorname{diag}(\hat{e}_{k|k-1})}{\phi}) \\ \times \operatorname{diag}\Big[\Phi_k^{22}(\Phi_k^{12})^{-1}\hat{e}_{k|k-1}\Big]^{-1}\Phi_k^{22}(\Phi_k^{12})^{-1} \end{aligned} \tag{6.33}$$

where γ_l is a $(n-m)\times(n-m)$ diagonal matrix. Overall, the GSVSF state update equation is as follows:

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k^{svsf}\hat{e}_{k|k-1} \\ &= \begin{bmatrix} \hat{x}_{k|k}^u \\ \hat{x}_{k|k}^l \end{bmatrix} = \begin{bmatrix} \hat{x}_{k|k-1}^u \\ \hat{x}_{k|k-1}^l \end{bmatrix} + \begin{bmatrix} K_k^u \\ K_k^l \end{bmatrix} \hat{e}_{k|k-1} \end{aligned} \tag{6.34}$$

and the a posteriori state error covariance matrix with fewer measurements than states is computed as:

$$\begin{aligned}\hat{P}_{k|k} = \hat{P}_{k|k-1} - \begin{bmatrix} K_k^u \\ K_k^l \end{bmatrix} H\hat{P}_{k|k-1} - \hat{P}_{k|k-1}H^T \begin{bmatrix} K_k^u \\ K_k^l \end{bmatrix}^T \\ + \begin{bmatrix} K_k^u \\ K_k^l \end{bmatrix} S_k \begin{bmatrix} K_k^u \\ K_k^l \end{bmatrix}^T\end{aligned} \tag{6.35}$$

where $S_k = H_1\hat{P}^{11}_{k|k-1}H_1^T + R_k$. Then the proposed Eq.(6.34) and Eq.(6.35) has included the measured and non-measured states. This strategy enables the iterative computation of the error covariance matrixes as given in Eq.(6.35) that is need for data association. The implementation of data associations are explained in the following sections. Figure 6.9 illustrates the visualization of our GSVSF tracking system.

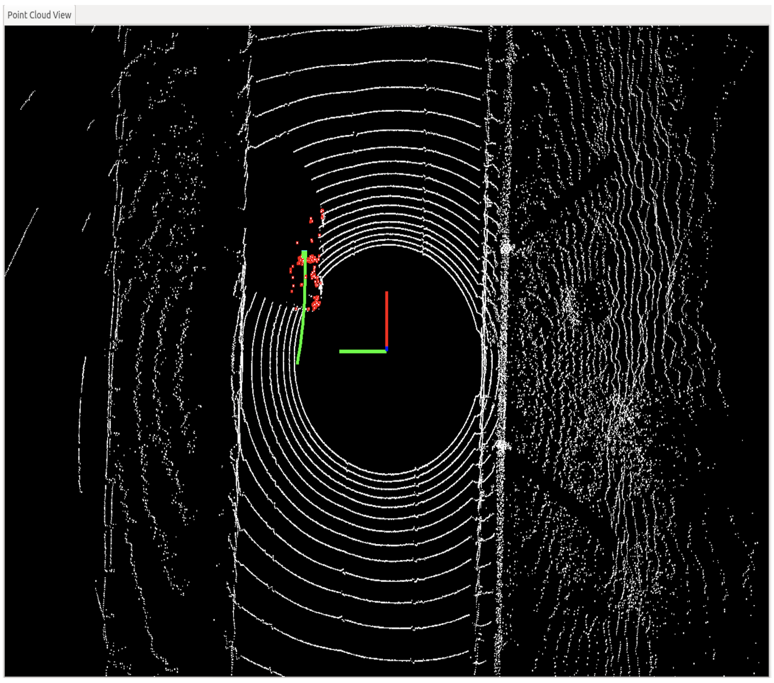
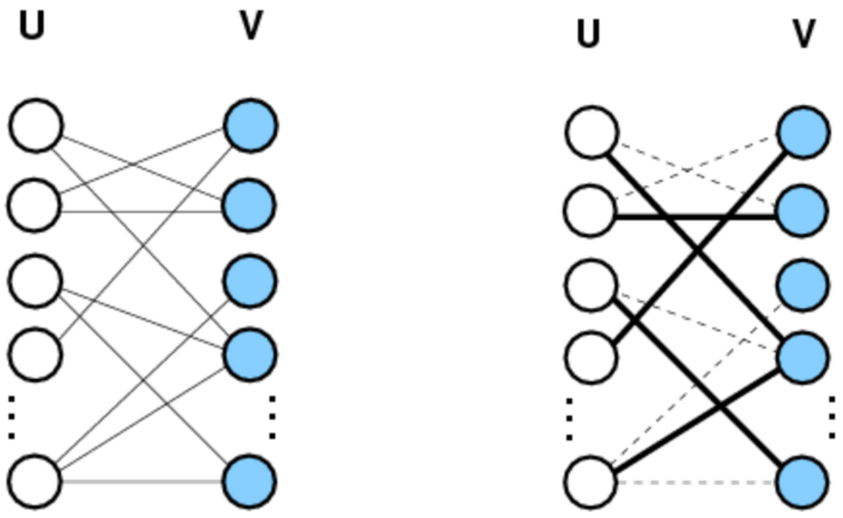


FIGURE 6.9: The visualization of our tracking system

6.4 Data Association Strategy

6.4.1 Hungarian Algorithm

In tracking targets with measurement in the presence of cluttered data, data association techniques are required to determine which of the received measurements are to be uses to update the trajectories of the targets. Suppose there are n tracks of vehicles and n observations of vehicles are available. The problems is how to assign each tracks to each observations so that the total cost is minimized. In our application, the cost is computed by using the *Euclidean distance* [**?**] between each track and each observation.



(A) $G = (V, U, E)$ (B) $\hat{G} = (V, U, \hat{E})$

FIGURE 6.10: The bipartite graph and its assignment

Consider $G = (V, U, E)$ be a bipartite and weighted graph as shown in Figure 6.10, where V and U are the sets of nodes, and n is the number nodes in each set so that $|V| = n$ and $|U| = n$. E is the set of edges. The edge weights may be stored in a cost matrix as $\vec{c}$ as shown

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix} \tag{6.36}$$

where c_{ij} is the cost of i^{th} track assigned to j^{th} observation. Because G is bipartite graph, V and U are two non-overlapping sets such that there are no edges with both endpoints in V and no edges with both endpoints in U. Therefore, the assignment problems is described as finding a min-weight matching in graph $\hat{G}$ as shown in Figure 6.10.

Assume the nodes from set U represents the number of tracks in our current track list and the nodes from set V represents the number of measurements at current frame. Let M_{ij} de defined the matching matches i^th node from U to j^{th} node from V. So mathematically the assignment problem associated with the min-weight matching problem is:

$$\min_{M_{i,j}} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} M_{ij} \tag{6.37}$$

$$M_{ij} = \begin{cases} 0, & \text{if } i^{th} \text{ track is not assigned the } j^{th} \text{ measurement} \\ 1, & \text{if } i^{th} \text{ track is assigned } j^{th} \text{ measurement} \end{cases}$$

subject to

$$\sum_{i=1}^{n} M_{ij} = 1$$
$$\sum_{j=1}^{n} M_{ij} = 1$$

Since this problem is a relaxation of the min-weight perfect matching problem, it follows that this vector corresponds to an optimal perfect matching. Thus, the next step is to find that matching. This is exactly the concept our algorithm takes.

One of the most well-known algorithms for solving the assignment problem is the Hungarian algorithm (HA) [92]. The HA involves the following four steps. The first two steps are executed once, while Steps 3 and 4 are repeated until an optimal assignment is found. The input of the algorithm is a n by n square matrix with only nonnegative entries. Consider an example where the set of track list $T = \{t_1, t_2, t_3, t_4\}$ need to matched by the set of measurements $O = \{o_1, o_2, o_3, o_4\}$. So the cost matrix C below shows the distance $c_{i,j}$ between the i^{th} track to the j^{th} measurement.

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = \begin{bmatrix} 8.2 & 8.3 & 6.9 & 9.2 \\ 7.7 & 3.7 & 4.9 & 9.2 \\ 1.1 & 6.9 & 0.5 & 8.6 \\ 0.8 & 0.9 & 9.8 & 2.3 \end{bmatrix} \tag{6.38}$$

where the distance are measurement by meter. Below the Hungarian algorithm is explained using this example.

1. **Subtract Row Minima:** For every row, find the lowest element and subtract this from every element in that row. For example, the smallest element in the first row is 6.9. So 6.9 is subtracted from each element in the first

row. Then the result of matrix C is:

$$C = \begin{bmatrix} 1.3 & 1.4 & 0 & 2.3 \\ 4.0 & 0 & 1.2 & 5.5 \\ 0.6 & 6.4 & 0 & 8.1 \\ 0 & 0.1 & 9.0 & 1.5 \end{bmatrix} \tag{6.39}$$

2. **Subtract Column Minima:** For every column, find the lowest element and subtract this from every element in that column. Similarly, the result of cost matrix is:

$$C = \begin{bmatrix} 1.3 & 1.4 & 0 & 0.8 \\ 4.0 & 0 & 1.2 & 4.0 \\ 0.6 & 6.4 & 0 & 6.6 \\ 0 & 0.1 & 9.0 & 0 \end{bmatrix} \tag{6.40}$$

3. **Cover All Zeros with a Minimum Number of Lines:** Draw lines through the row and columns that have the 0 entries such that the fewest lines possible are drawn.

   - If there are n lines drawn, an optimal assignment of zeros is possible and the algorithm is finished.
   - If the number of lines is less than n, then the optimal number of zeros is not yet reached. Go to the next step.

$$C = \begin{bmatrix} 1.3 & 1.4 & 0 & 0.8 \\ 4.0 & 0 & 1.2 & 4.0 \\ 0.6 & 6.4 & 0 & 6.6 \\ 0 & 0.1 & 9.0 & 0 \end{bmatrix}$$

Because the number of lines required is 3 as shown in above matrix, which is lower than the size of the matrix (n=4), so Step 4 is processed.

4. **Create Additional Zeros:** Find the smallest entry not covered by any line. Subtract this entry from each row that is not crossed out, and then add it to each column that is crossed out. The result of the cost matrix is:

$$C = \begin{bmatrix} 0.7 & 0.8 & 0 & 0.2 \\ 4.0 & 0 & 1.8 & 4.0 \\ 0 & 5.8 & 0 & 6.0 \\ 0 & 0.1 & 9.6 & 0 \end{bmatrix} \tag{6.41}$$

Then, go back to Step 3.

The minimum number of lines required to cover all zeros in the matrix is now determined by 4 lines:

$$C = \begin{bmatrix} \cancel{0.7} & \cancel{0.8} & \cancel{0} & \cancel{0.2} \\ \cancel{4.0} & \cancel{0} & \cancel{1.8} & \cancel{4.0} \\ \cancel{0} & \cancel{5.8} & \cancel{0} & \cancel{6.0} \\ \cancel{0} & \cancel{0.1} & \cancel{9.6} & \cancel{0} \end{bmatrix}$$

Because the number of required lines is 4, which is equals the size of the matrix (n=4), the algorithm stops. Therefore, the following zeros cover an optimal assignment:

$$C = \begin{bmatrix} 0.7 & 0.8 & \mathbf{0} & 0.2 \\ 4.0 & \mathbf{0} & 1.8 & 4.0 \\ \mathbf{0} & 5.8 & 0 & 6.0 \\ 0 & 0.1 & 9.6 & \mathbf{0} \end{bmatrix} \tag{6.42}$$

This corresponds to the following optimal assignment in the original cost matrix:

$$C = \begin{bmatrix} 8.2 & 8.3 & \mathbf{6.9} & 9.2 \\ 7.7 & \mathbf{3.7} & 4.9 & 9.2 \\ \mathbf{1.1} & 6.9 & 0.5 & 8.6 \\ 0.8 & 0.9 & 9.8 & \mathbf{2.3} \end{bmatrix} \tag{6.43}$$

Therefore, track t_1 is associated to measurement o_3, track t_2 is associated to measurement o_2, track t_3 is associated to measurement o_1, and track t_4 is associated to measurement o_4. The total distance of this optimal assignment is to 6.9 + 3.7 + 1.1 + 2.3 = 14.

6.4.2 Probabilistic Data Association Filter

The probabilistic data association filter (PDAF) uses a Bayesian approach to compute the association probabilities by weighting the influence of the various candidate measurements. Instead of choosing the nearest neighbor as the case in HA, the standard PDAF is to evaluates each measurement that falls in a gate around the predicted measurement [93]. As for PDAF, there are generally three assumptions [94]:

- only one target of interest is present.
- no more than one measurement can originate from a target.
- the track has been initialized.
- the past information about the target is approximated by

$$Pr\{x_k|z_{1:k}\} = \aleph\{x_k; \hat{x}_{k|k-1}, \hat{P}_{k|k-1}\} \tag{6.44}$$

where $\aleph\{x_k; \hat{x}_{k|k-1}, P_{k|k-1}\}$ is the the normal probability density function with argument x_k, mean $x_{k|k-1}$ and covariance matrix $\hat{P}_{k|k-1}$.

To avoid searching the entire measurement set for the measurements originated from a specific target, an ellipsoidal gate is set up for each target, and such a gate is called a validation region as shown in Figure 6.11.

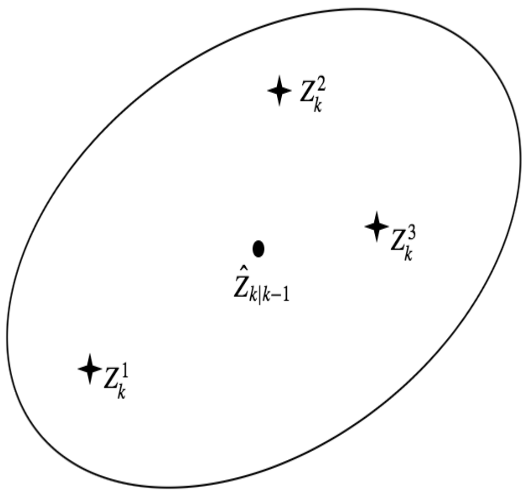


FIGURE 6.11: The ellipsoidal validation region of a target centered at its predicted measurement $\hat{z}_{k|k-1}$ represented by dot. Three measurements have fallen within the validation region represented by star.

The probability which the target-originated measurement falls within the validation region is called the gate probability Pr^G [97]. If more than one measurement falls within the gate, then an association uncertainty arises. It is required to decide which measurement is originated form the target and therefore should be used to update the track.

The validation region R_k is defined as follows:

$$R_k = \{[z_k - \hat{z}_{k|k-1}]^T S_k^{-1} [z_k - \hat{z}_{k|k-1}] \leq r_g\} \tag{6.45}$$

where r_g is the gate threshold and S_k is the covariance of the innovation term.

Suppose the set of measurements at time k is

$$\boldsymbol{Z_k} = \{z_k^i\}_{i=1}^m = \{z_k^1, z_k^2, ..., z_k^m\} \tag{6.46}$$

where m represents the number of measurements at time k. So the cumulative set of measurements through time k is as below.

$$\boldsymbol{Z_{1:k}} = \{\boldsymbol{Z_1}, \boldsymbol{Z_2}, ..., \boldsymbol{Z_k}\} \tag{6.47}$$

In addition, suppose the association event is denoted as:

$$\Im_k^i = \begin{cases} z_k^i \text{ is the measurement originated} & i = 1, 2, ..., m \\ = \text{no correct measurement is present} & i = 0 \end{cases} \tag{6.48}$$

By using the *total probability theory* [95] with respect to Eq.(6.44), the conditional mean of the state at time k can be written as

$$\begin{aligned} \hat{x}_{k|k} &= \mathbb{E}\{x_k|\boldsymbol{Z_{1:k}}\} = \sum_{i=0}^{m} \mathbb{E}\{x_k|\Im_k^i, \boldsymbol{Z_{1:k}}\} Pr\{\Im_k^i|\boldsymbol{Z_{1:k}}\} \\ &= \sum_{i=1}^{m} \hat{x}_{k|k}^i \beta_k^i \end{aligned} \tag{6.49}$$

where $\hat{x}_{k|k}^i$ is the updated state conditioned on the event that the i^{th} measurement is correct and β_k^i is the association probability. Let the probability of the target detections occur independently over time be Pr^D. Thus β_k^i is obtain from the PDA with the Poisson clutter model [96] as below:

$$\beta_k^i = \begin{cases} \dfrac{\pounds_k^i}{1 - Pr^D Pr^G + \sum_{i=1}^m \pounds_k^i} & i = 1, 2, ..., m \\ \dfrac{1 - Pr^D Pr^G}{1 - Pr^D Pr^G + \sum_{i=1}^m \pounds_k^i} & i = 0 \end{cases} \tag{6.50}$$

where Pr^G is the gate probability, and:

$$\pounds_k^i = \aleph\{z_k^i; \hat{z}_{k|k-1}, S_k\} \tag{6.51}$$

is the likelihood ratio of the measurement z_k^i being originated from target.

As the association events are mutually exclusive $\sum_{i=0}^m \beta_k^i = 1$. Then the innovation term is given by:

$$I_k^i = (\hat{e}_{k|k-1})^i = z_k^i - H\hat{x}_{k|k-1} \tag{6.52}$$

Then the corresponding total combined innovation term is:

$$\hat{E}_{k|k-1} = \sum_{i=1}^m \beta_k^i (\hat{e}_{k|k-1})^i \tag{6.53}$$

Hence, the estimation equation is expressed by:

$$\hat{x}_{k|k} = \sum_{i=0}^{m} \beta_k^i \hat{x}_{k|k}^i = \hat{x}_{k|k-1} + K_k \hat{E}_{k|k-1} \tag{6.54}$$

As one can see from the above equations, if there are no measurement falling into the validation region, namely $i = 0$, the state estimation is simply updated by state prediction $\hat{x}_{k|k}^0 = \hat{x}_{k|k-1}$. The error covariance associated with the updated state estimation can be expressed as follows:

$$\hat{P}_{k|k} = \hat{P}_{k|k-1}\beta_k^0 + [1 - \beta_k^0]\hat{P}_{k|k}^c + \tilde{P}_k \tag{6.55}$$

where $\hat{P}_{k|k}^c$ represents the standard state estimation error covariance, as below:

$$\hat{P}_{k|k}^c = (\mathbb{I} - K_k H)\hat{P}_{k|k-1} \tag{6.56}$$

and $\tilde{P}_k$ captures the effect of uncertain associations:

$$\tilde{P}_k = K_k[\sum_{i=1}^{m} \beta_k^i I_k^i (I_k^i)^T - \hat{E}_{k|k-1}(\hat{E}_{k|k-1})^T](K_k)^T \tag{6.57}$$

6.5 Proposed GSVSF-HA/PDAF Approach

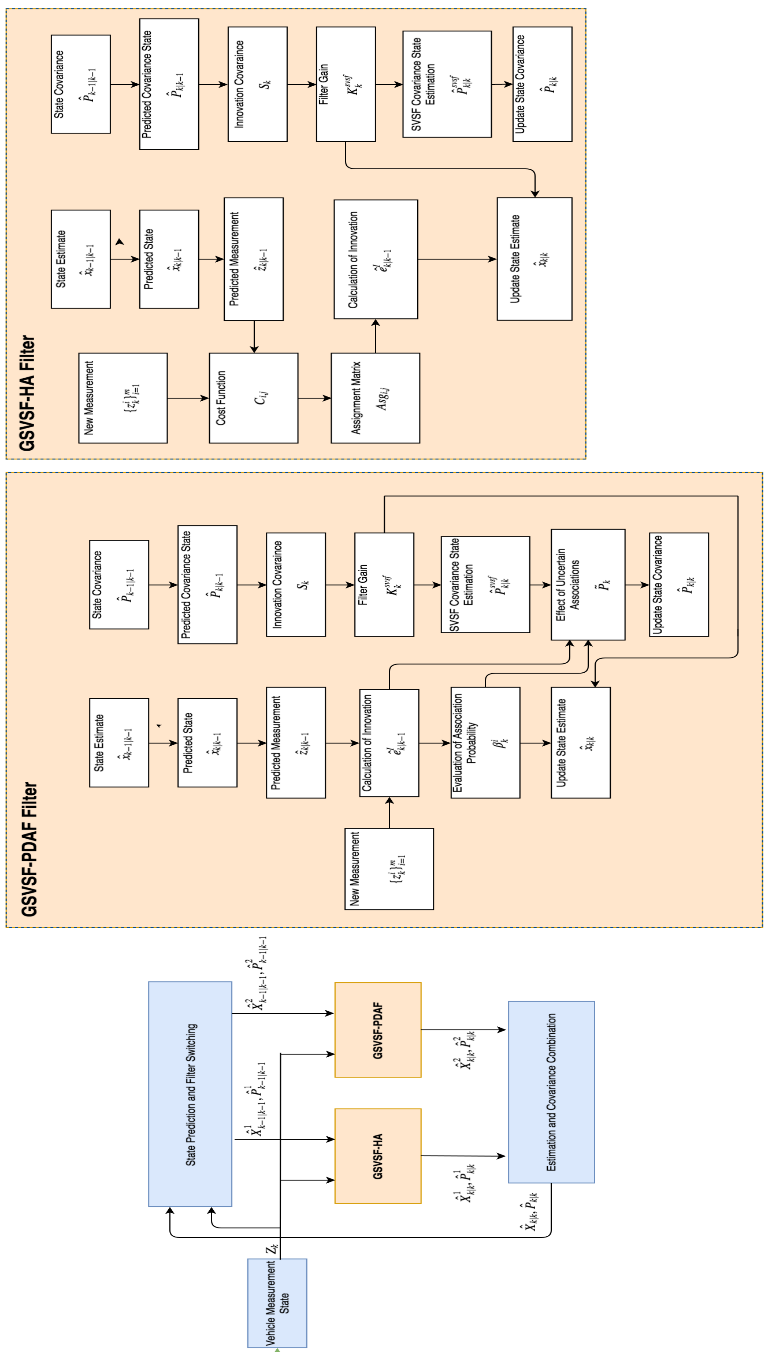


FIGURE 6.12: The data flow of proposed GSVSF-HA/PDAF approach

As presented in Figure 6.12, the proposed GSVSF-HA/PDAF strategy makes use of two filters: (i) GSVSF-HA, and (ii) GSVSF-PDAF. These two filters are explained in the following sections.

6.5.1 GSVSF-HA

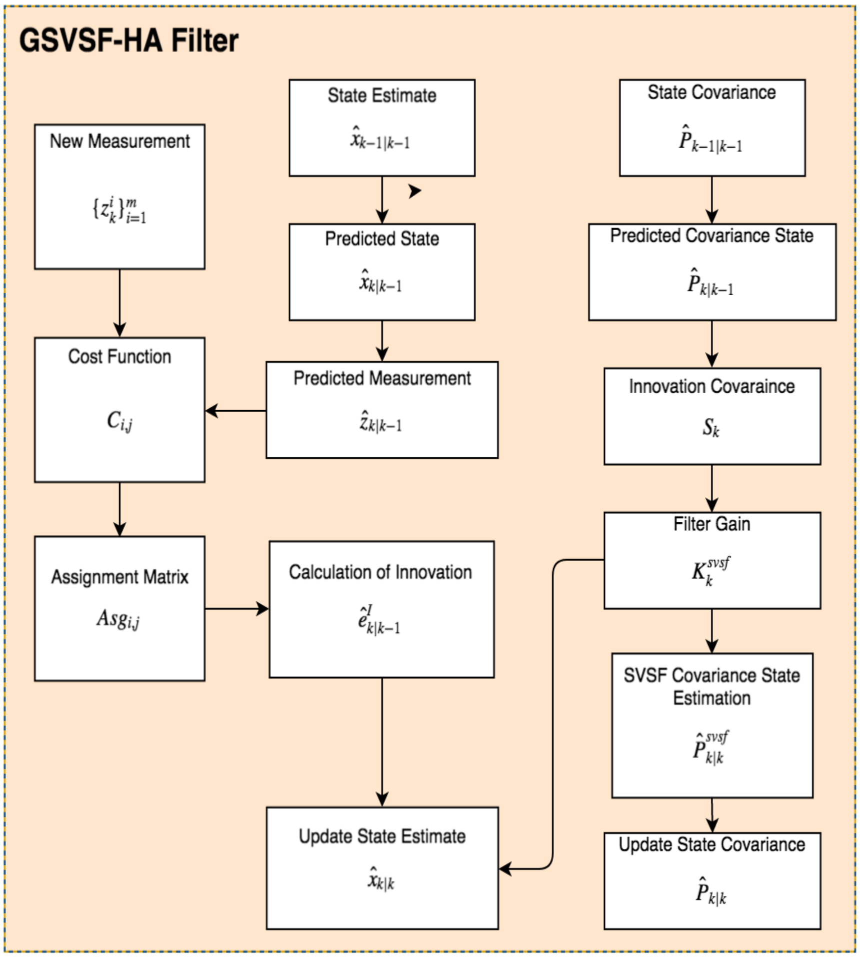

GSVSF-HA Filter
State Estimate $\hat{x}_{k-1|k-1}$
State Covariance $\hat{P}_{k-1|k-1}$
New Measurement $\{z_k^i\}_{i=1}^m$
Predicted State $\hat{x}_{k|k-1}$
Predicted Covariance State $\hat{P}_{k|k-1}$
Cost Function $C_{i,j}$
Predicted Measurement $\hat{z}_{k|k-1}$
Innovation Covaraince S_k
Filter Gain K_k^{svsf}
Assignment Matrix $Asg_{i,j}$
Calculation of Innovation $\hat{e}_{k|k-1}^I$
SVSF Covariance State Estimation $\hat{P}_{k|k}^{svsf}$
Update State Estimate $\hat{x}_{k|k}$
Update State Covariance $\hat{P}_{k|k}$

FIGURE 6.13: The GSVSF-HA filter

Figure 6.13 shows the flowchart of GSVSF-HA Filter. The proposed GSVSF-HA algorithm consisted of the following steps:

1. **Prediction**

   In the first step, the predicted state and measurements are computed by using the system and measurement models. Thus the a priori state estimates are calculated.

$$\begin{aligned} \hat{x}^1_{k|k-1} &= f(\hat{x}^1_{k-1|k-1}, u_k) \\ \hat{z}^1_{k|k-1} &= h(\hat{x}^1_{k|k-1}, v_k) \\ \hat{P}^1_{k|k-1} &= A\hat{P}^1_{k-1|k-1}A^T + Q_k \end{aligned} \tag{6.58}$$

2. **Measurement Matching**

   The cost function $C_{i,j}$ simply depends on the *Euclidean distance* between the i^{th} track and the j^{th} measurement where $0 < i < n$ and $0 < j < m$ as shown from Eq. (6.36) :

$$C_{i,j} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots\,\vdots & & \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{bmatrix} \tag{6.59}$$

3. **Data Association**

   Let's define the assignment matrix $Asg_{i,j} \in \{0, 1\}$, and the optimal assignment in the cost matrix is computed by satisfying the following formulations

$$\begin{aligned} &\min_{Asg_{i,j}} \sum_{i=1}^{n} Asg_{i,j} C_{i,j}. \\ &\text{subject to} \sum_{j=1}^{m} Asg_{i,j} = 1; \quad \sum_{i=1}^{n} Asg_{i,j} = 1 \end{aligned} \tag{6.60}$$

Additional checks are performed to reject the large distance. So the assignment matrix is modified by setting $Asg_{i,j} = 0$ if and only if $C_{i,j} > r$, where r is the threshold distance (i.e. 5m) between i^{th} track and the j^{th} measurement.

4. **Update**

The measurement errors are computed based on the assignment matrix generated from the HA:

$$\hat{e}_{k|k-1} = z_k - H\hat{z}^{1}_{k|k-1} \tag{6.61}$$

Then the SVSF corrective gain is calculated by Eq.(6.33). Thus the state estimates are updated corresponding to Eq. (6.34) and Eq.(6.35) respectively:

$$\begin{aligned} &\hat{x}^{1}_{k|k} = \hat{x}^{1}_{k|k-1} + K^{svsf}_{k}\hat{e}_{k|k-1} \\ &\hat{P}^{1}_{k|k} = \hat{P}^{1}_{k|k-1} - K^{svsf}_{k} H \hat{P}^{1}_{k|k-1} - \hat{P}^{1}_{k|k-1} H^{T} (K^{svsf}_{k})^{T} \\ &+ K^{svsf}_{k} S_{k} (K^{svsf}_{k})^{T} \end{aligned} \tag{6.62}$$

Therefore the state estimates are updated, Step 1-4 will be repeated when the new set of measurements being obtained.

6.5.2 GSVSF-PDAF

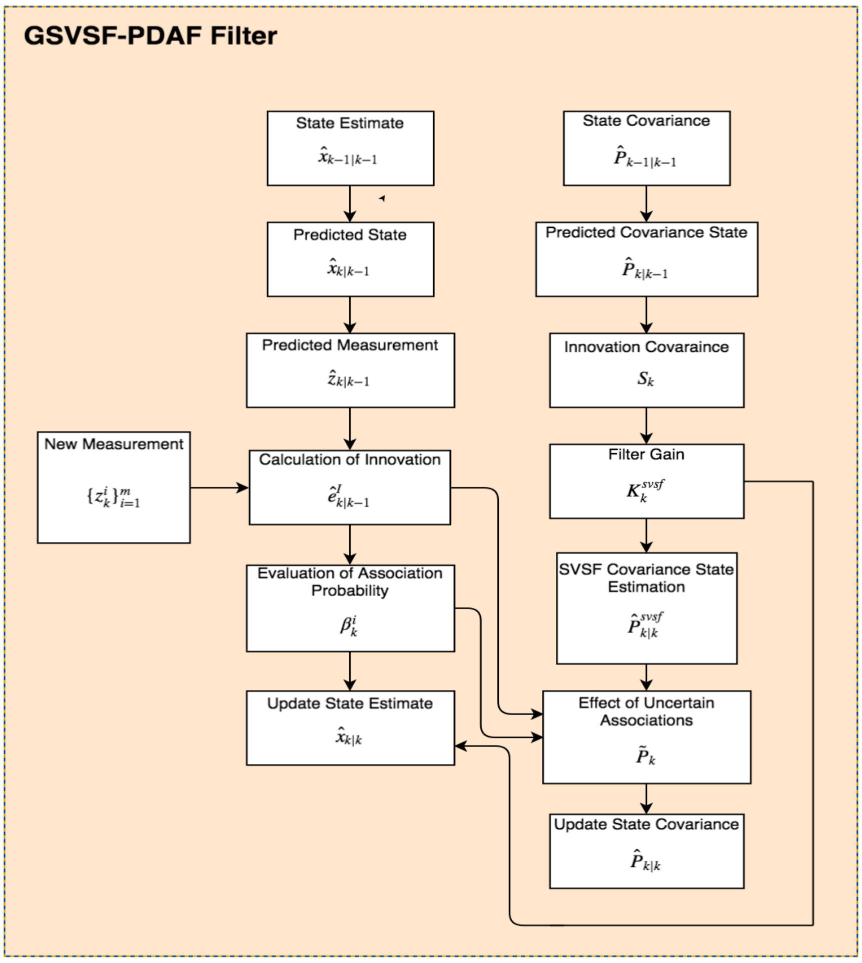

GSVSF-PDAF Filter
State Estimate $\hat{x}_{k-1|k-1}$
State Covariance $\hat{P}_{k-1|k-1}$
Predicted State $\hat{x}_{k|k-1}$
Predicted Covariance State $\hat{P}_{k|k-1}$
Predicted Measurement $\hat{z}_{k|k-1}$
Innovation Covaraince S_k
New Measurement $\{z_k^i\}_{i=1}^m$
Calculation of Innovation $\tilde{e}_{k|k-1}^i$
Filter Gain K_k^{svsf}
Evaluation of Association Probability β_k^i
SVSF Covariance State Estimation $\hat{P}_{k|k}^{svsf}$
Update State Estimate $\hat{x}_{k|k}$
Effect of Uncertain Associations $\tilde{P}_k$
Update State Covariance $\hat{P}_{k|k}$

FIGURE 6.14: The GSVSF-PDAF filter

The second GSVSF based tracking and data association filter is proposed in this subsection. Figure 6.14 demonstrates the flowchart of GSVSF-PDAF filter. The proposed GSVSF-PDAF algorithm is as followings:

1. **Prediction**

   Within the PDAF algorithm, the state vector, the measurement and the state error covariance are predicted:

$$\begin{aligned}\hat{x}^2_{k|k-1} &= f(\hat{x}^2_{k-1|k-1}, u_k) + w_k \\ \hat{z}^2_{k|k-1} &= H\hat{x}^2_{k|k-1} + v_k \\ \hat{P}^2_{k|k-1} &= A\hat{P}^2_{k-1|k-1}A^T + Q_k\end{aligned} \tag{6.63}$$

   The innovation covariance matrix corresponding to the correct measurement is also computed

$$S_k = H\hat{P}^2_{k|k-1}H^T + R_k \tag{6.64}$$

2. **Measurement Matching**

   The validation gate equation from Eq.(6.45) is set up for each time step to determine the candidate measurements for association :

$$R_k = \{[z_k - \hat{z}^2_{k|k-1}]^T S_k^{-1}[z_k - \hat{z}^2_{k|k-1}] \leq r_g\} \tag{6.65}$$

   where r_g is the gate threshold. Thus the volume of the gate is given by [94]

$$\mathrm{Vol}^{R_k} = C_{n_z} r_g^{\frac{n_z}{2}} |S_k|^{\frac{1}{2}} \tag{6.66}$$

   where n_z is the dimension of the measurement and c_{n_z} is the volume of the n_z-dimensional unit hypersphere. For example, $c_1 = 2, c_2 = \pi$ and $c_3 = \frac{4\pi}{3}$.

3. **Data Association**

   The association probabilities β_k^i and the combined innovation term $\hat{E}_{k|k-1}$ are computed by Eq.(6.48) and Eq.(6.53), respectively.

4. **Update**

   The state update equation of PDAF is

$$\hat{x}^2_{k|k} = \hat{x}^2_{k|k-1} + K_k^{svsf} \hat{E}_{k|k-1} \tag{6.67}$$

   where K_k^{svsf} is the corrective gain from Eq.(6.31). Corresponding to [99] the SVSF covariance matrix is computed by:

$$\begin{aligned} \hat{P}^{svsf}_{k|k} &= [\mathbb{I} - K_k^{svsf} H]\hat{P}_{k|k-1}[\mathbb{I} - K_k^{svsf} H]^T \\ &+ K_k^{svsf} R_k (K_k^{svsf})^T \end{aligned} \tag{6.68}$$

   The updated state estimation covariance is calculated by:

$$\hat{P}^2_{k|k} = \beta_k^0 \hat{P}^2_{k|k-1} + [\mathbb{I} - \beta_k^0]\hat{P}^{svsf}_{k|k} + \tilde{P}_k \tag{6.69}$$

   where $\tilde{P}_k$ is the effect of uncertain associations. The the posteriori measurement error is obtained by:

$$\hat{E}_{k|k} = [\mathbb{I} - H K_k^{svsf}]\hat{E}_{k|k-1} \tag{6.70}$$

Therefore the state estimates are updated, Step 1-4 will be repeated when the new set of measurements being obtained.

6.5.3 GSVSF-HA/PDAF

Overall, the steps of proposed method GSVSF-HA/PDAF consists of four steps:

1. **State and Measurement Predictions**: Given the set of state estimate $\hat{\boldsymbol{X}}_{k-1|k-1}$ from previous time step, the predicted measurement is computed as:

$$\begin{aligned}\hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_k) \\ \hat{z}_{k|k-1} &= H\hat{x}_{k|k-1} + v_k\end{aligned} \tag{6.71}$$

2. **Gating for Filter Selection**: Then the distance ℓ_k between from the predicted state $\hat{z}_{k|k-1}$ to the measurement z_k for each target is defined as a stochastic distance as:

$$\ell_k = [z_k - \hat{z}_{k|k-1}]^T S_k^{-1} [z_k - \hat{z}_{k|k-1}] \tag{6.72}$$

where S_k is the covariance of the innovation term. Let r_g be the gate threshold. Then the set of states for GSVSF-HA and the set of states for GSVSF-PDAF is specified by the following two conditions:

   (a) At current time step k, if the measurements are located within the gate region, $\ell_k < r_g$, then the data assassination strategy is implemented by PDAF. Therefore, the GSVSF-PDAF filter is used to update the state estimate.

(b) At current time step k, if the measurements are located outside the gate region, $\ell_k > r_g$, then the data assassination strategy is implemented by HA. Therefore, the GSVSF-HA filter is used to update the state estimate.

3. **Filtering Processing**: The two filters: GSVSF-HA and GSVSF-PDAF described from previous subsections are processed independently to update the state estimates. The sets of updated state estimates of the two filters are $\hat{\boldsymbol{X}}^1_{k|k}$ and $\hat{\boldsymbol{X}}^2_{k|k}$, and the The sets of covariance matrix of the two filters are $\hat{\boldsymbol{P}}^1_{k|k}$ and $\hat{\boldsymbol{P}}^2_{k|k}$, respectively.

4. **Combination**: The overall state estimate $\hat{\boldsymbol{X}}_{k|k}$ and the covariance $\hat{\boldsymbol{P}}_{k|k}$ are combined from the results of state estimation of two filters:

$$\begin{aligned}\hat{\boldsymbol{X}}_{k|k} &= \hat{\boldsymbol{X}}^1_{k|k} \cup \hat{\boldsymbol{X}}^2_{k|k} \\ \hat{\boldsymbol{P}}_{k|k} &= \hat{\boldsymbol{P}}^1_{k|k} \cup \hat{\boldsymbol{P}}^2_{k|k}\end{aligned} \tag{6.73}$$

6.6 Experimental Result

Table 6.2 illustrates the parameter values as used by our proposed methods where n is the length of state vector and m is the length of measurement vector:

A series of experiments were performed to verify the robustness and effectiveness of the proposed method. The data were collected by our real time system which is described in Chapter 2. The proposed methods are evaluated using three scenarios

GSVSF-HA/PDAF		
Method	GSVSF-HA	GSVSF-PDAF
Process noise covariance	0.5*eye(n)	0.5*eye(n)
Measurement noise covariance	0.5*eye(m)	0.5*eye(m)
SVSF Gain Parameter	0.1	0.1
SVSF boundary layer Width	15	15
Gate threshold	N/A	4.61
Gate probability	N/A	0.9
Detection probability	N/A	0.98

Table 6.2: The parameters of proposed methods

that spanning different scenarios from heavy traffic to light traffic as shown in Figure 6.15. The heavy traffic scenario consists of much more targets than light traffic scenarios. There are 20% less traffic between high and medium traffic while there are 80% less traffic between medium and light traffic. The comparative study also includes Kalman Filter based HA and Kalman filter based PDA. The following tables show the evaluated performances of the proposed methods in the three scenarios where σ represents the variance of RMSEs of the state estimates. In total, the RMSEs of state estimation process are computed over 1900 frames of run, for the three different scenarios: light traffic, medium traffic, and heavy traffic. Because our real time data were collected while driving on the highway, the motion dynamics of vehicles were mostly in the y-direction and hence modeling uncertainty is larger in the y direction. Therefore, the RMSE in y direction is the key factor for modeling uncertainties.

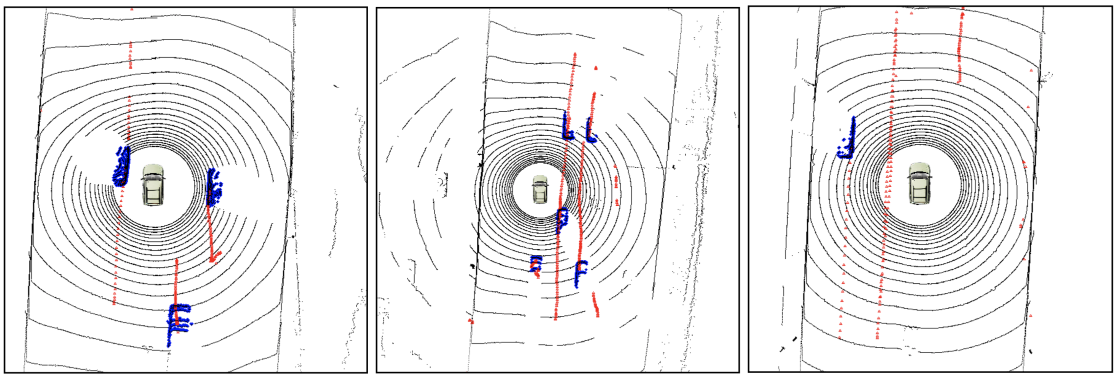

(A) Scenario 1 (B) Scenario 2 (C) Scenario 3

FIGURE 6.15: Different experimental scenarios

Method	GSVSF-HA/PDAF	GSVSF-PDAF	GSVSF-HA	KF-PDAF	KF-HA
Scenarios 1					
$RMSE$	**0.2414**	0.2610	0.2940	0.2839	0.3764
$\sigma(RMSE)$	0.0664	0.0770	0.0212	0.0182	0.0282
Scenarios 2					
$RMSE$	**0.1318**	0.1595	0.3124	0.2971	0.3932
$\sigma(RMSE)$	0.0409	0.0801	0.0792	0.1900	0.0398
Scenarios 3					
$RMSE$	**0.2390**	0.2410	0.2440	0.2447	0.2855
$\sigma(RMSE)$	0.0606	0.0602	0.0614	0.0982	0.0813

TABLE 6.3: The performance of state estimation

Due to the robust characteristics of Generalized SVSF, the RMSE of GSVSF based methods are considerable lower than KF based methods. More specifically, in Scenario 1, the RMSE of GSVSF-PDAF is 8.06 % lower than KF-PDAF, and the RMSE of GSVSF-HA is 21.8% lower than KF-HA. In Scenario 2, the RMSE of GSVSF-PDAF is 26.1 % lower than KF-PDAF, and the RMSE of GSVSF-HA is 20.5% lower than KF-HA. In Scenario 3, the RMSE of GSVSF-PDAF is 1.51 % lower than KF-PDAF, and the RMSE of GSVSF-HA is 14.5% lower than KF-HA.

For data association strategy, the RMSE of our proposed HA/PDAF based method is lower than either PDAF or HA based methods. From Scenarios 1 of Table 6.3, the RMSEs of GSVSF-PDAF/HA is 7.5 % lower than GSVSF-PDAF and 17.89% than GSVSF-HA. In Scenario 2, the RMSEs of GSVSF-PDAF/HA is 15.95% lower than GSVSF-PDAF and 57.81% than GSVSF-HA. In Scenario 3, where the data of the traffic is less than the previous two scenarios, the RMSEs of GSVSF-PDAF/HA is comparable to GSVSF-PDAF and GSVSF-HA.

The HA algorithm considers all measurements and uses only the nearest one to update the state, whereas PDAF uses a weighted sum of all of measurements within each gate to update the state. So, the performance of HA deteriorates in the case of multiple targets located closely, i.e., in the heavy traffic scenario as shown in Figure 6.15(B). This is evident from the results in Table 6.3 where the performance of GSVSF-HA and KF-HA are the worst amongst the scompared methods.

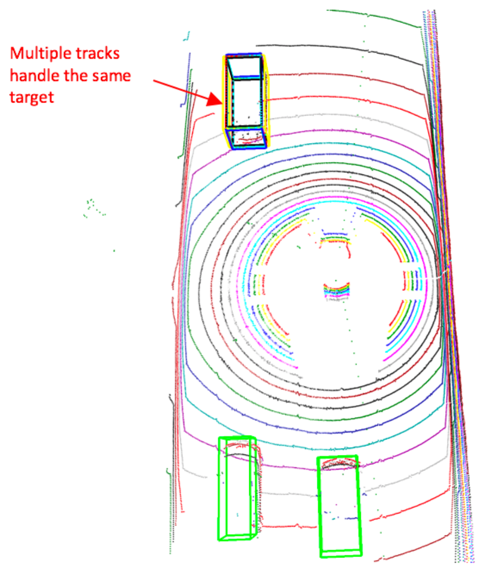
Table 6.4 shows he number of correctly tracked targets(CT), the number of falsely tracked targets(FT), the rate of true positives (TP), the rate of false positives (FP), and the computation time:

Figure 6.16(A) illustrates the multiple tracks to same target problem that leads to increase computation time and reduced accuracy. However, our proposed GSVSF-HA/PDAF method has shown to reduce the computation time while

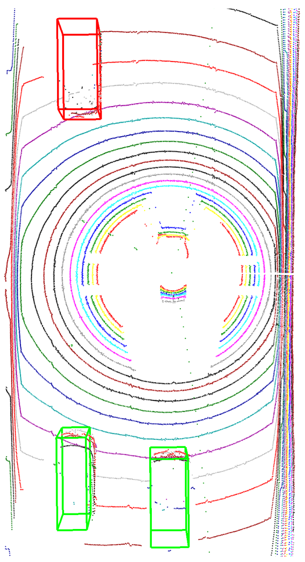
Scenarios 1 Total Frames: 690 Total Vehicle: 63					
Method	GSVSF-HA/PDAF	GSVSF-PDAF	GSVSF-HA	KF-PDAF	KF-HA
CT	59	59	56	59	56
FT	4	4	7	4	7
TP	**93.65%**	93.65%	88.88%	93.65%	88.88%
FP	6.35%	6.35%	11.12%	6.35%	11.12%
Time(ms)	**13.22**	23.84	16.23	18.13	12.21
Scenarios 2 Total Frames: 702 Total Vehicle: 79					
Method	GSVSF-HA/PDAF	GSVSF-PDAF	GSVSF-HA	KF-PDAF	KF-HA
CT	75	75	72	72	70
FT	4	4	7	7	9
TP	**94.9%**	94.9%	91.13%	91.13 %	88.6%
FP	5.1%	5.1%	8.87%	8.87%	11.4%
Time(ms)	**15.12**	25.74	19.43	20.86	12.342
Scenarios 3 Total Frames: 546 Total Vehicle: 8					
Method	GSVSF-HA/PDAF	GSVSF-PDAF	GSVSF-HA	KF-PDAF	KF-HA
CT	8	8	8	8	8
FT	0	0	0	0	0
TP	**100%**	100%	100%	100%	100%
FP	0%	0%	0%	0%	0%
Time(ms)	**10.43**	13.10	12.53	11.13	7.242

TABLE 6.4: The performance evaluation of trackers

maintaining acceptable and comparable performance amongst methods as shown in Table 6.4 and Figure 6.16(B).



(A) PDAF based methods



(B) Scenario 2

FIGURE 6.16: GSVSF-HA/PDAF method

6.7 Conclusion

In summary, this chapter presents a generalized form of SVSF integrated with two different data association algorithms, referred to as GSVSF-HA/PDAF. To investigate the performance of the proposed method, a comparative analysis is performed to evaluate the robustness and effectiveness of the method against other popular tracking methods and data association methods in different real time scenarios. To evaluate the performance of the tracker, the proposed method is also investigated in terms of CT, FT, TP, FT and computation time and compared with other methods. The proposed algorithm has demonstrated to be a more reliable tracking algorithm for real time applications, based on the comparison criteria. The SVSF-based method benefits from the robustness of the SVSF strategy. Therefore, it is able to reduce the effects of modeling uncertainties as evident by comparing the RMSE in state estimation results. Because the robustness of the PDAF-based method degrades when gating failure exist in the association between the targets and the measurements.Therefore, our proposed hybrid method is developed to address this problem while still maintaining the effectiveness for real time application. In order to further monitor the dynamic movements of surrounding vehicles, the future work should be focused on developing behavioral patterns for vehicles as symbolic representations of context-dependent motion primitives that a vehicle is able to conduct.

Chapter 7

Overview of Real Time System and its Implementation

7.1 Introduction

In this chapter, the overview of the real time system and its implementation is presented. The overall data flow is shown in Figure 7.1. The entire system is divided into seven main modules. Each module contains several task-specific sub-modules. The modular structure allows for flexible construction for specific tasks by configuring and including an appropriate set of modules.

For flexibility, our developed system is decomposed into several modules. One of our main object is to enable the flexible construction of system by composing existing sets of modules. Because dependencies between modules mean that one module cannot be activated and executed without the other, dependencies are

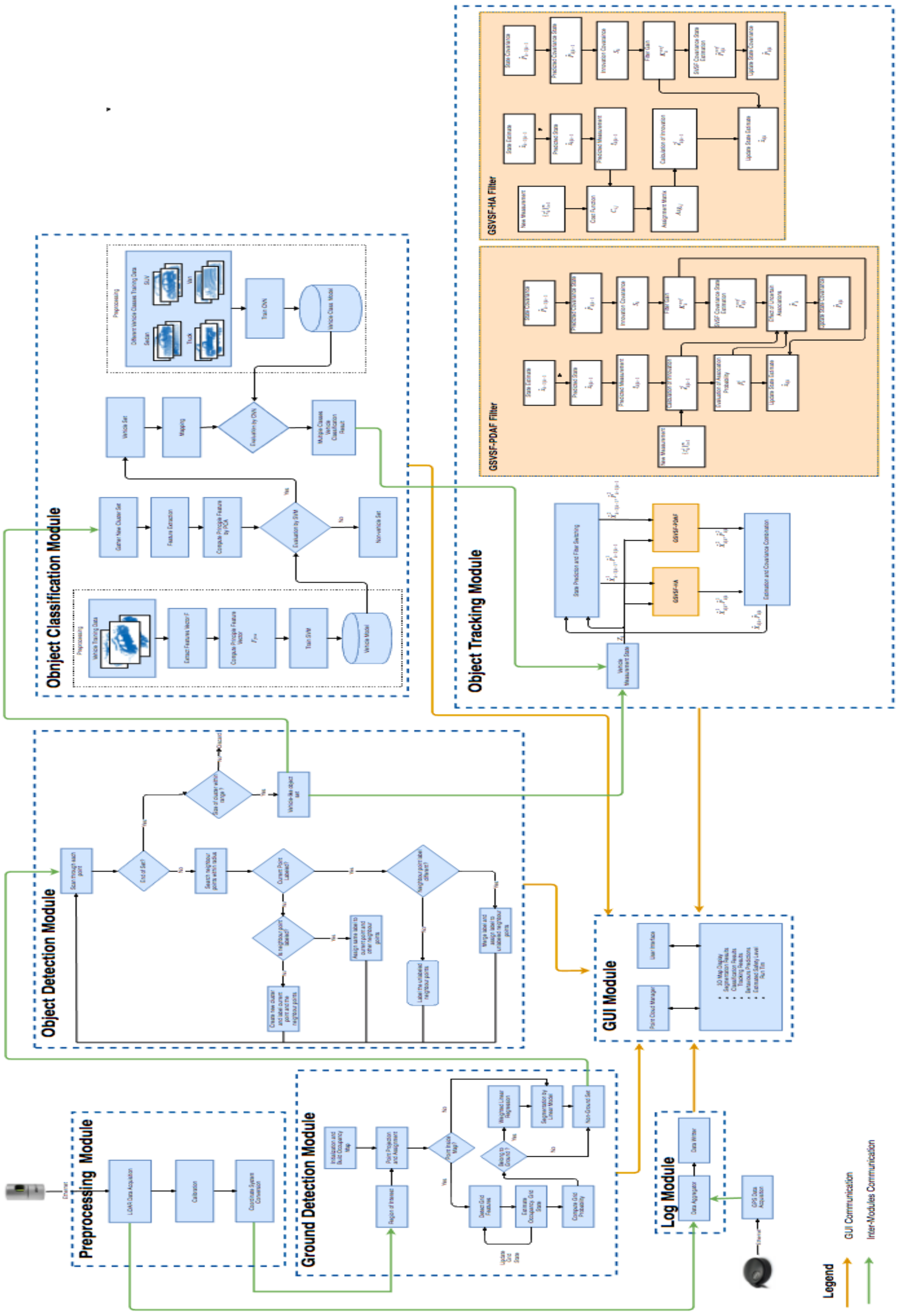


FIGURE 7.1: The overall real time system

needed to be minimized to the extent possible. In the next section, the implementation of the system structure is described.

7.2 Modules

7.2.1 Preprocessing Module

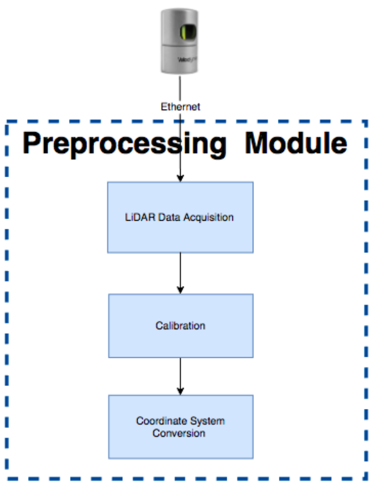
Module Name	Preprocessing
Input	LiDAR raw data
Output	A set of points in Cartesian coordinate system
Processing Time	average 5ms
Description	The preprocessing module is designed to read different calibration files for different types of LiDAR sensor which is described in Chapter 2. The LiDAR raw data uses the polar coordinate system to represent the three dimensional space. Therefore, this module can also convert the polar coordinate system to Cartesian coordinate system [98].
Flowchart	 Ethernet Preprocessing Module LiDAR Data Acquisition Calibration Coordinate System Conversion

Table 7.1: The preprocessing module

7.2.2 Ground Detection Module

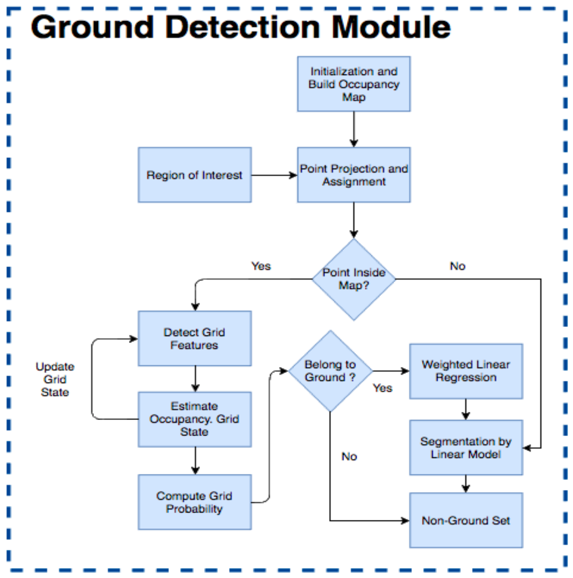
Module Name	Ground Detection
Input	A set of points in Cartesian coordinate system
Output	A set of points labeled as “Non-ground” or “Ground”
Processing Time	average 1.5ms
Description	The ground detection module provides efficient and robust estimates of the ground position and slope by taking advantage of the geometry of a roof mounted LiDAR sensor. The design of the ground detection module is described in Chapter 3.
Flowchart	 Ground Detection Module Initialization and Build Occupancy Map Region of Interest Point Projection and Assignment Yes Point Inside Map? No Detect Grid Features Update Grid State Estimate Occupancy. Grid State Compute Grid Probability Belong to Ground ? Yes No Weighted Linear Regression Segmentation by Linear Model Non-Ground Set

Table 7.2: The ground detection module

7.2.3 Object Detection Module

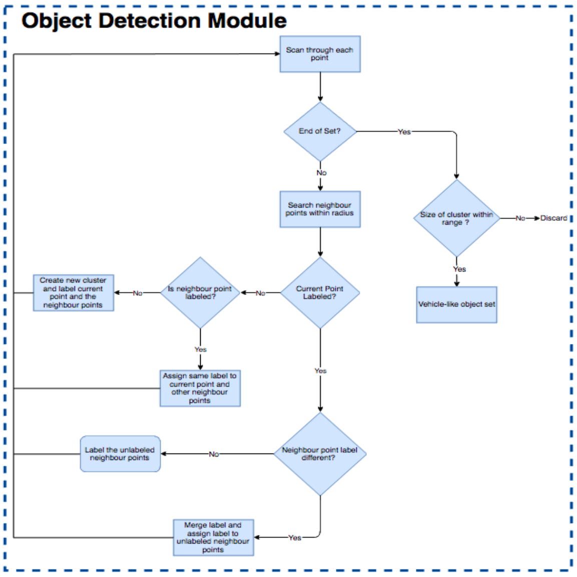
Module Name	Object Detection
Input	A set of points labeled as ether "Ground" or "Non-ground"
Output	A set of clusters
Processing Time	average 55ms
Description	The object detection module is to examine a collection of non-ground points, and group the points into clusters according to *Euclidean distance* as presented in Chapter 4. The threshold value of distance for separating clusters is a trade-off between accuracy and computation time. In addition, the size of the clusters is constrained by a predefined 3D bounding box in order to get rid of large objects or small objects which are usually considered as debris.
Flowchart	 Object Detection Module Scan through each point End of Set? Yes No Search neighbour points within radius Size of cluster within range ? No Discard Yes Vehicle-like object set Current Point Labeled? No Is neighbour point labeled? No Create new cluster and label current point and the neighbour points Yes Assign same label to current point and other neighbour points Yes Neighbour point label different? No Label the unlabeled neighbour points Yes Merge label and assign label to unlabeled neighbour points

TABLE 7.3: The object detection module

7.2.4 Object Classification Module

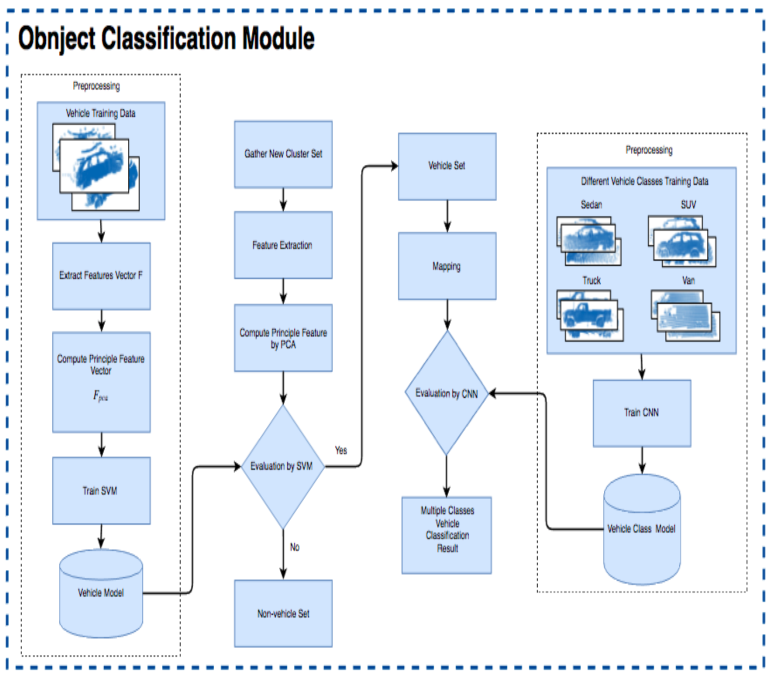
Module Name	Object Classification
Input	A set of clusters
Output	A set of data labeled as “Sedan”, “SUV”, “Van”, and “Truck”
Processing Time	training time: 6 hours, and classification time: 100ms
Description	The object classification module is presented in Chapter 5. Any cluster be recognized and classified is considered as non-vehicle object. The classification result can be used to determine parameters for high level control systems such as autonomous driving.
Flowchart	 Obnject Classification Module Preprocessing Vehicle Training Data Extract Features Vector F Compute Principle Feature Vector F_{pca} Train SVM Vehicle Model Gather New Cluster Set Feature Extraction Compute Principle Feature by PCA Evaluation by SVM Yes No Non-vehicle Set Vehicle Set Mapping Evaluation by CNN Multiple Classes Vehicle Classification Result Preprocessing Different Vehicle Classes Training Data Sedan SUV Truck Van Train CNN Vehicle Class Model

TABLE 7.4: The object classification module

7.2.5 Object Tracking Module

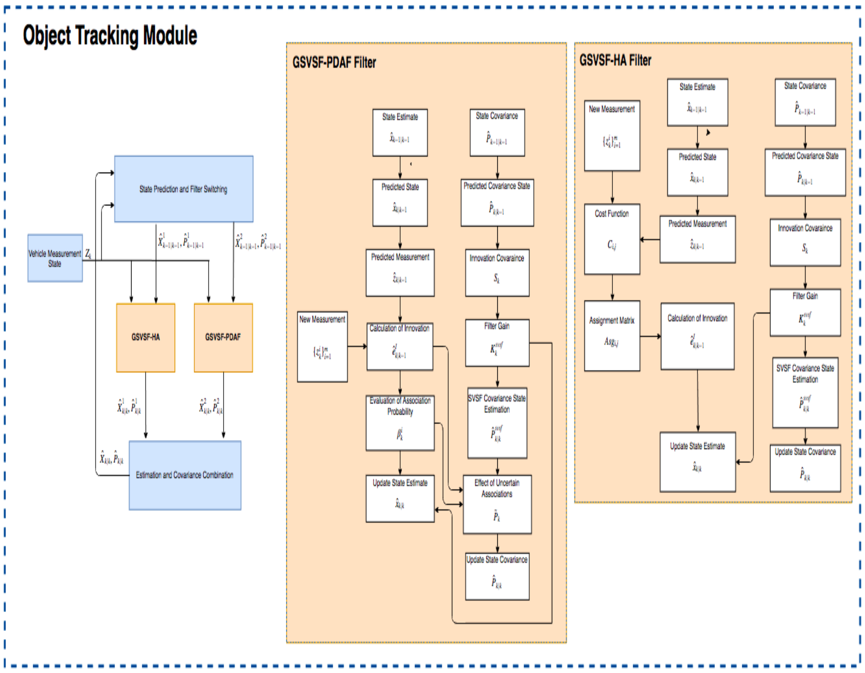
Module Name	Object Tracking
Input	A set of data labeled as “Sedan”, “SUV”, “Van”, and “Truck”
Output	A set of state estimates: position, velocity, and turn rate
Processing Time	average 12.9ms
Description	Given the classification results, the object tracking module is used to analysis the motion state of tracked objects, as described in Chapter 7. The estimated motion states such as position and velocity can be extended for future work
Flowchart	 Object Tracking Module State Prediction and Filter Switching Vehicle Measurement State Z_k $\hat{X}^1_{k-1\|k-1}, \hat{P}^1_{k-1\|k-1}$ $\hat{X}^2_{k-1\|k-1}, \hat{P}^2_{k-1\|k-1}$ GSVSF-HA GSVSF-PDAF $\hat{X}^1_{k\|k}, \hat{P}^1_{k\|k}$ $\hat{X}^2_{k\|k}, \hat{P}^2_{k\|k}$ $\hat{X}_{k\|k}, \hat{P}_{k\|k}$ Estimation and Covariance Combination GSVSF-PDAF Filter State Estimate $\hat{x}_{k-1\|k-1}$ State Covariance $\hat{P}_{k-1\|k-1}$ Predicted State $\hat{x}_{k\|k-1}$ Predicted Covariance State $\hat{P}_{k\|k-1}$ Predicted Measurement $\hat{z}_{k\|k-1}$ Innovation Covaraince S_k New Measurement $\{z^i_k\}^m_{i=1}$ Calculation of Innovation $\hat{e}^j_{k\|k-1}$ Filter Gain K^{svsf}_k Evaluation of Association Probability β^i_k SVSF Covariance State Estimation $\hat{P}^{svsf}_{k\|k}$ Update State Estimate $\hat{x}_{k\|k}$ Effect of Uncertain Associations $\tilde{P}_k$ Update State Covariance $\hat{P}_{k\|k}$ GSVSF-HA Filter New Measurement $\{z^i_k\}^m_{i=1}$ State Estimate $\hat{x}_{k-1\|k-1}$ State Covariance $\hat{P}_{k-1\|k-1}$ Predicted State $\hat{x}_{k\|k-1}$ Predicted Covariance State $\hat{P}_{k\|k-1}$ Cost Function $C_{i,j}$ Predicted Measurement $\hat{z}_{k\|k-1}$ Innovation Covaraince S_k Filter Gain K^{svsf}_k Assignment Matrix $Asg_{i,j}$ Calculation of Innovation $\hat{e}^j_{k\|k-1}$ SVSF Covariance State Estimation $\hat{P}^{svsf}_{k\|k}$ Update State Estimate $\hat{x}_{k\|k}$ Update State Covariance $\hat{P}_{k\|k}$

Table 7.5: The object tracking module

7.2.6 Graphical User Interface (GUI) Module

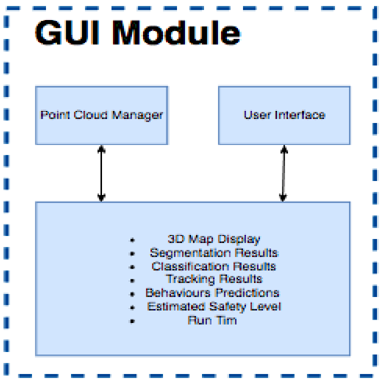
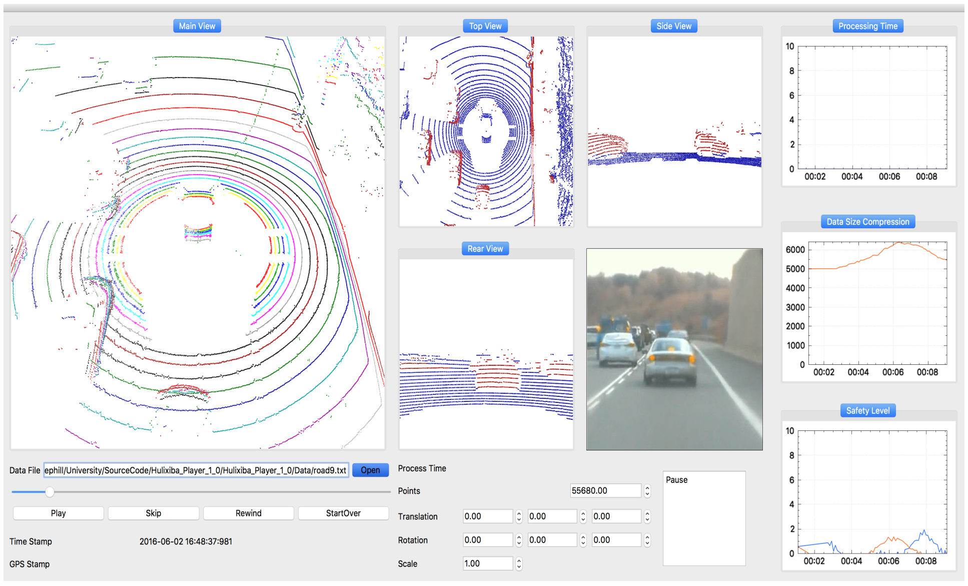
Module Name	GUI
Input	• A set of points in Cartesian coordinate system • A set of points labeled as ether "Ground" or "Non-ground" • A set of clusters • A set of data labeled as "Sedan", "SUV", "Van", and "Truck" • A set of state estimates: position, velocity, and turn rate
Output	Visual representation of data
Processing Time	average 26ms
Description	The graphical user interface (GUI) Module is to visually represent the results from other modules as shown in Figure 7.2.
Flowchart	

TABLE 7.6: The graphical user interface module



Data File ephill/University/SourceCode/Hulixiba_Player_1_0/Hulixiba_Player_1_0/Data/road9.txt

Time Stamp 2016-06-02 16:48:37:981

GPS Stamp

Process Time

Points 55680.00

Translation 0.00 0.00 0.00

Rotation 0.00 0.00 0.00

Scale 1.00

FIGURE 7.2: The GUI main window

7.3 Inter-Module Communication

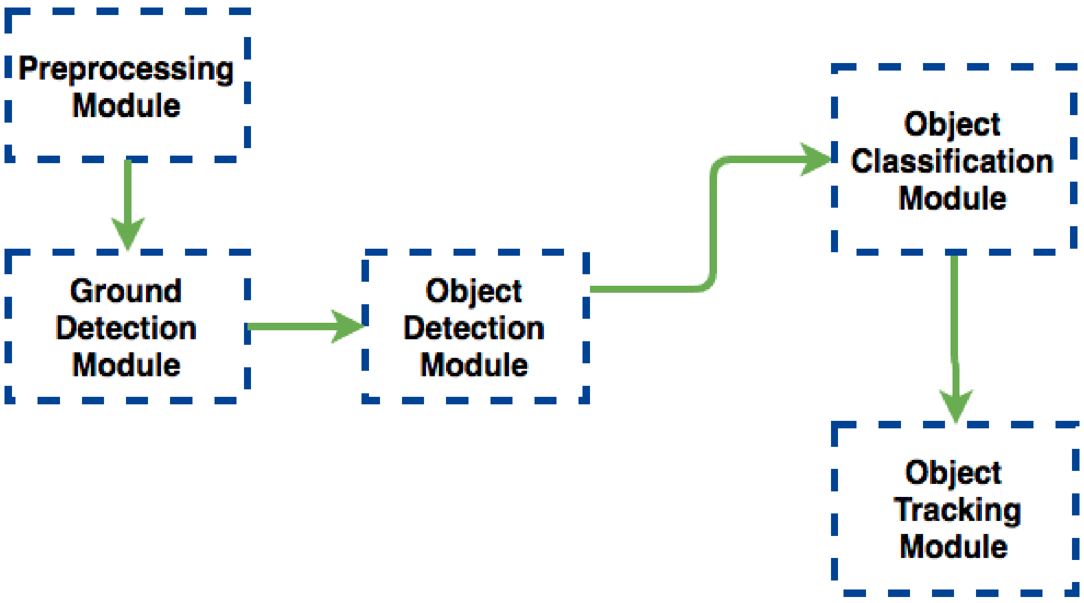


FIGURE 7.3: Example of initial data flow diagram

Figure 7.3 illustrates an example of the initial data flow diagram where each module follows the result from previous module. However, the overall computation time by this initial data flow design is not meeting our real time criteria. Because the object classification module usually consume a lot of time to process due to the complex neural network, so the object tracking module is being inactive until the object classification module has finished its processing. Therefore, the inter-module communication has been modified such that each module is no longer depending on the previous module. Instead, modules are decoupled by using publish-subscribe communication mechanism [101].

In the publish-subscribe communication, modules which are sending data are called publishers, and modules which are receiving the data are called subscribers. A publisher can send data to multiple subscribers, and a subscribers can receive data from multiple publishers. However, a publisher has no direct link to subscribers. The interactions between publishers and subscribers are controlled by data brokers as shown in Figure 7.4. Therefore, data is sent from publishers to the data broker, and then the data broker forwards the data to the subscribers.

Figure 7.4 show an example of the modified data flow diagram. The Object Detection Module publishes Data A to the Data Broker. Since both the Object Classification Module and the Object Tracking Module have subscribed to the Data Broker, then each module receiver a copy of Data A from the Data Broker.

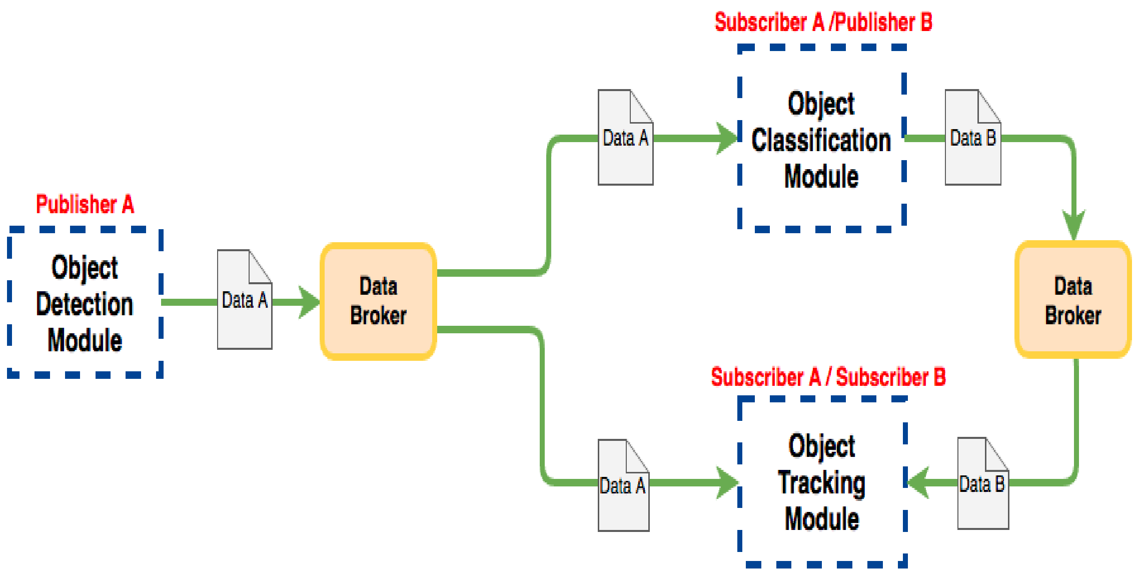


FIGURE 7.4: The inter-modules communication

Continuing with the example of Figure 7.4, the Object Tracking Module is being processed whenever a new set of clusters is sent from the Object Detection Module. The set of classification data published by Object Classification Module is used to manger the track list from the tracking system. Because the Data Broker is designed such that If a subscriber is under processing, the broker does not retain data for it. Therefore, frames lost for Object Classification Module is unavoidable. In our application, the average number of frames lost is about 3 frames per iteration, which has minimum effects on the Object Tracking Module.

In order for users or high level control systems to immediately know the results from each module such as ground points or positions of targets, a GUI Module is also designed by using publish-subscribe mechanism for data sharing. Figure 7.5 show an example of the modified GUI communication. The Ground Detection Module send a set of points labeled as "Non-ground" or "Ground" to the

Object Detection module and GUI Module. The high level systems can get this information for control operation from the GUI Module without waiting for other modules.

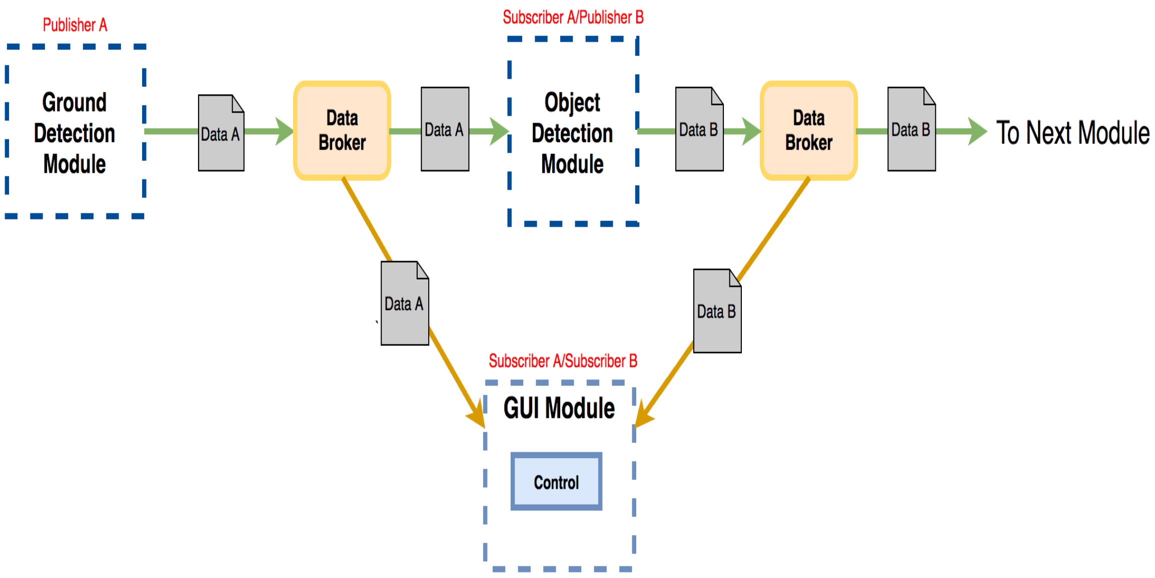
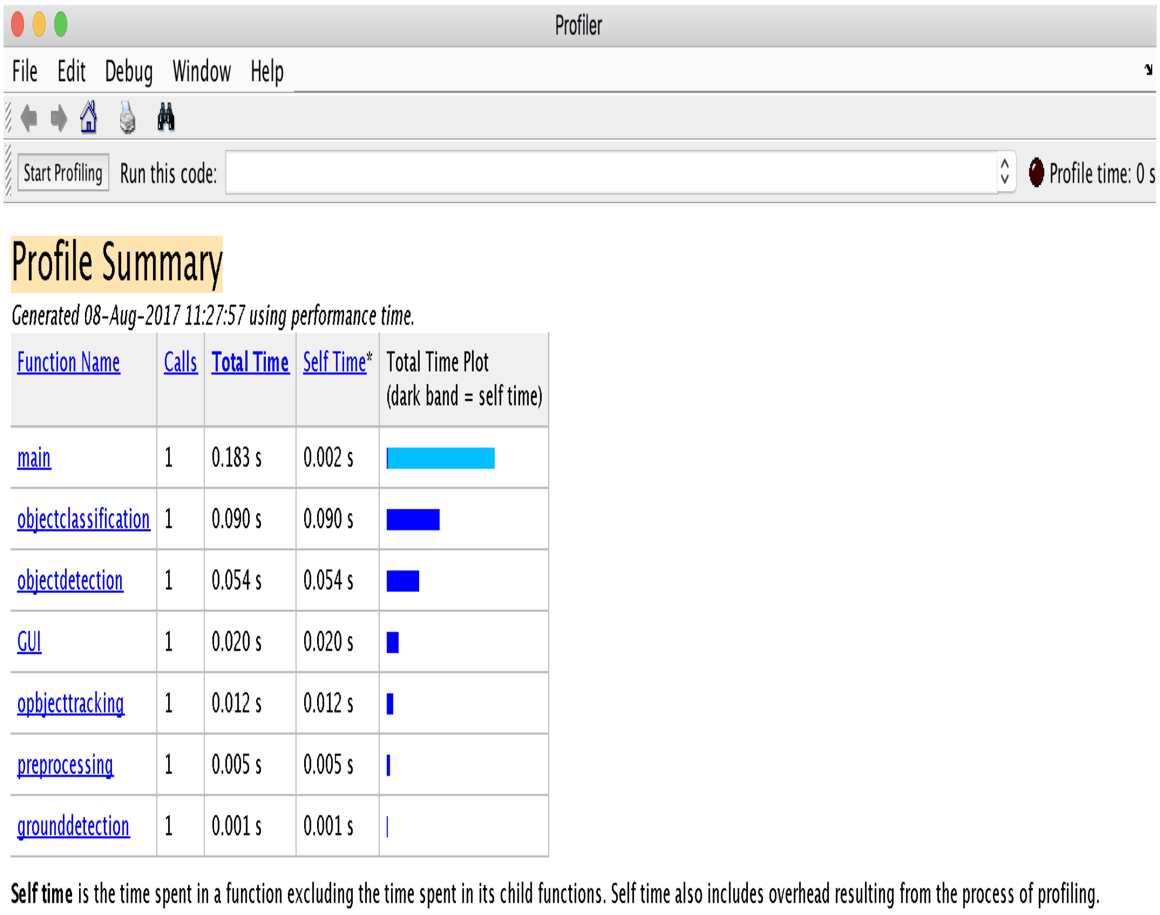


FIGURE 7.5: The GUI communication

7.4 Matlab Development

Matlab is a computer program that provides the user with a convenient environment for rapid prototyping. The Matlab profiler [102] provides a way to measure the program execution time. After identifying which functions are consuming the most time, the possible performance improvements were evaluated. Knowing the execution time of the code could help us to debug and optimize it.



Profile Summary

Generated 08-Aug-2017 11:27:57 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	0.183 s	0.002 s	
objectclassification	1	0.090 s	0.090 s	
objectdetection	1	0.054 s	0.054 s	
GUI	1	0.020 s	0.020 s	
opbjecttracking	1	0.012 s	0.012 s	
preprocessing	1	0.005 s	0.005 s	
grounddetection	1	0.001 s	0.001 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Figure 7.6: The Matlab profiler report

Using the Matlab to explore the proposed system structure decides how to optimize algorithms. Matlab profiler was used to see how long of each module are needed to execute. The profiler will indicate the bottleneck areas and how they should be modified for developing a more efficient algorithm.

One of application based on our proposed algorithm is shown at Figure 7.7. There are 4 panels in this figure. On the bottom left panel, the front view camera recording the flow of traffic is displayed.The top left window shows the raw LiDAR

3D point cloud. The top right frame shows the perception of surrounding of our host vehicle. The bottom right panel shows the results of our tracking algorithm. This display has many innovated features, for example, it can identify vehicles going over the speed limit as indicated in the bottom right window by using a red square.

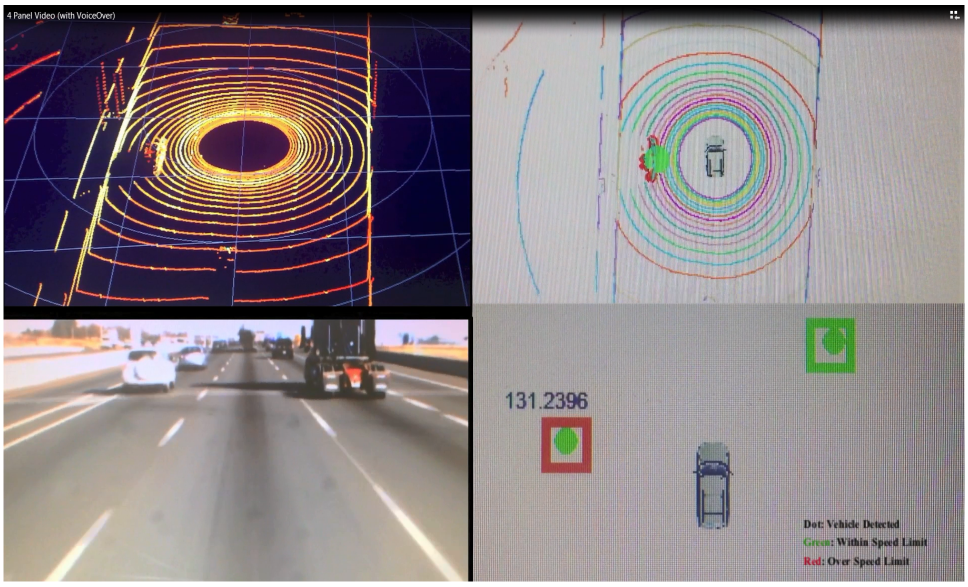
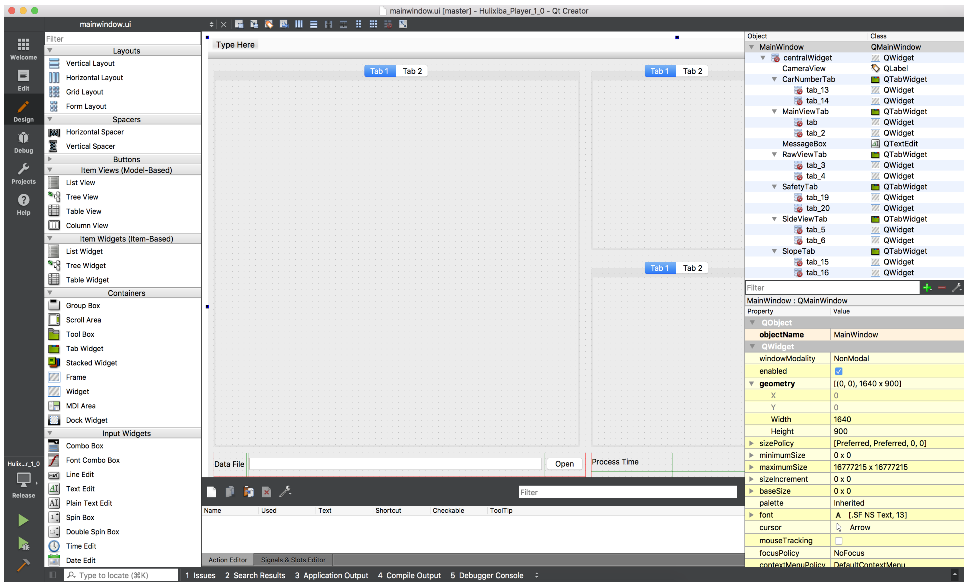


FIGURE 7.7: One of our applications developed by Matlab

7.5 C++ Development



Object	Class
MainWindow	QMainWindow
centralWidget	QWidget
CameraView	QLabel
CarNumberTab	QTabWidget
tab_13	QWidget
tab_14	QWidget
MainViewTab	QTabWidget
tab	QWidget
tab_2	QWidget
MessageBox	QTextEdit
RawViewTab	QTabWidget
tab_3	QWidget
tab_4	QWidget
SafetyTab	QTabWidget
tab_19	QWidget
tab_20	QWidget
SideViewTab	QTabWidget
tab_5	QWidget
tab_6	QWidget
SlopeTab	QTabWidget
tab_15	QWidget
tab_16	QWidget

MainWindow : QMainWindow

Property	Value
QObject	
objectName	MainWindow
QWidget	
windowModality	NonModal
enabled	✓
geometry	[(0, 0), 1640 x 900]
X	0
Y	0
Width	1640
Height	900
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [.SF NS Text, 13]
cursor	Arrow
mouseTracking	☐
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu

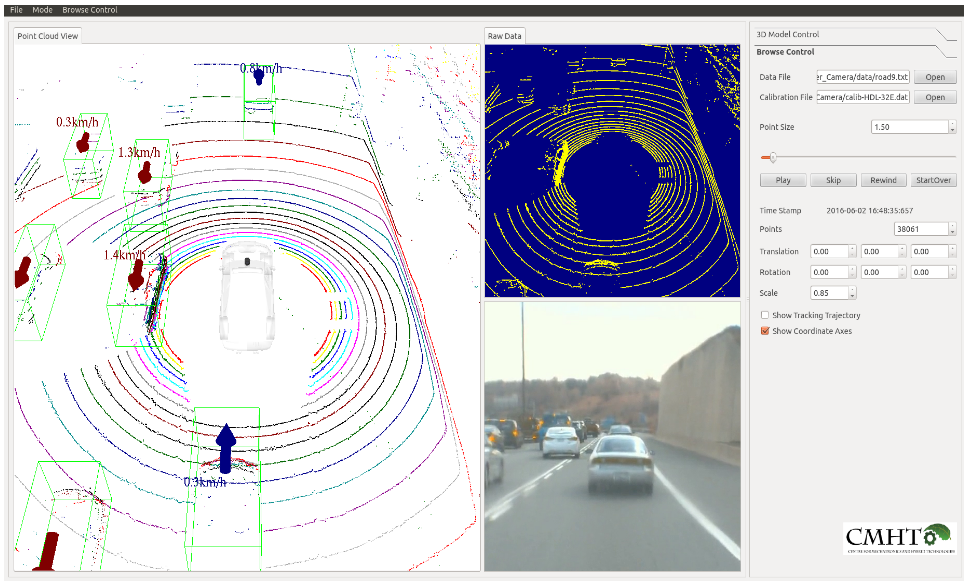
FIGURE 7.8: Qt creator framework

Fast and reliable communications is a basic requirement in almost all modern applications, but real time systems take it to extreme and require real time responses from the network. In particular, our LiDAR uses a lean protocol over the UDP transport layer. Because UDP does not need to retransmit lost packets, sending data incurs less delay. So the UDP-based network communication provides for extremely fast message transmission. Therefore, C++ has an outstanding performance for fast and efficient implementation of communication strategies. Therefore, all code from Matlab was re-implemented to C++ in order to operate under a real time environment.

Writing in C++ also enables great control, making the possibility of working with open source libraries such as the STL, Boost and so on [103]. The code is compiled to native binaries that will run at full speed without the need for a virtual machine.

Qt Creator was implemented as an Integrated Development Environment (IDE) for our GUI application development because of its easy GUI programming. The main reason for selecting Qt is that the rendering framework is vector based, thus allowing multiple composition modes.

Figure 7.7 demonstrated our real time implementation of developed LiDAR perception system by using C++.



Data File er_Camera/data/road9.txt
Calibration File Camera/calib-HDL-32E.dat
Point Size 1.50
Time Stamp 2016-06-02 16:48:35:657
Points 38061
Translation 0.00 0.00 0.00
Rotation 0.00 0.00 0.00
Scale 0.85
☐ Show Tracking Trajectory
☑ Show Coordinate Axes

FIGURE 7.9: The real time implementation by C++

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This chapter discusses the main conclusions and contributions derived from the research, followed by recommendations for future work that can complement this work.

In this research, a real time LiDAR perception system was developed for autonomous vehicle application. Being prerequisites for implementation of autonomous vehicle, the LiDAR perception system plays a significant role for decision making. Keeping with the goal to achieve a fast, robust and reliable solution to the perception problem, our approach has been presented, validated, and demonstrated by real time implementation. The summarized conclusion of each chapter is presented in the following.

Chapter 3 presented a probability occupancy grid map based approach for ground segmentation to address the issues of over-segmentation, under-segmentation and slow-segmentation because of non-flat surfaces. Ground segmentation is a necessary intermediate step to partition using data into several sets with different requirements. Experiments on real-life traffic data have shown that our proposed algorithms can robustly and efficiently perform a real-time ground segmentation in different scenarios.

With the ground segmentation algorithm presented in Chapter 3, Chapter 4 proposed an algorithm that is efficiently segments a given 3D point cloud using a Radially Bounded Nearest Neighbor (RBNN) method while maintain its ability for real time processing based on the static *Kd-tree*. The proposed algorithm was tested on different urban traffic scenarios and the evaluations showed comparable results against the clustering technique DBDCAN, but over 50% lower in computation time.

A supervised learning approach based on our LiDAR sensor for vehicle type classification is proposed in Chapter 5. The proposed approach, which is referred as PCA-SVM-CNN, was trained and tested on our real time LiDAR system. The real time experimental results had demonstrated the effectiveness and robustness of the proposed method.

Chapter 6 presented a generalized form of SVSF integrated with two different data association algorithms, referred to as GSVSF-HA/PDAF. A comparative analysis is performed to evaluate the robustness and effectiveness against other tracking methods. Based on the comparison results, The proposed algorithm has demonstrated to be a more reliable tracking algorithm while still maintaining its feasibility for real time application.

In Chapter 7, our overall structure of our real time system using the LiDAR sensor was presented. The publish-subscribe communication was used to decouple the modules. This publish-subscribe communication structure allows for flexibility construction by subscribing the appropriate modules for specific requirements. In order for high level control systems to immediately know the results from each module, a GUI Module is also designed by using publish-subscribe mechanism for data sharing. MATLAB profiler was used to measure the program execution time and decide how to optimize algorithms while C++ was used for implementation of our system for real time application

This thesis presents a real time robust and efficient LiDAR based perception system for autonomous vehicle applications. The design of ground detection, object detection, object classification, and object tracking had demonstrated the effectiveness and robustness in real time dynamics environment . The proposed framework provides a structure for building autonomous vehicles.

8.2 Future work

In this section, possible future work directions are discussed for our intelligent vehicle application

8.2.1 Behaviors Prediction

Inexperienced drivers are not as good as experienced drivers in predicting what will happen next in a driving scene. So behavior prediction of other drivers would provide more time for planning and reaction for intelligent vehicles. In order to anticipate how the driving scene is going to evolve, a prediction algorithm is needed. An intelligent vehicle equipped with a behavior perdition system can warn its driver if another vehicle is likely to influence its behavior. Therefore, the driver can avoid potential dangerous situation. For this reason, one of the strategies to utilize is the Interacting Multiple Models (IMM) [100] to represent different behaviors. Another approach would be to combine machine learning techniques for estimating trajectories associated with behaviors.

8.2.2 Integration with Other Sensors

The developed algorithms are designed to be expandable with multiple sensors. The camera is the first choice to be integrated with our LiDAR perception system. It could be used for developing lane detection algorithms. In addition, analyzing thermal images taken with an infrared thermal camera can help detecting vehicles robustly under poor conditions such as snow and thick fog.

Appendix A

Supplementary

A1 LiDAR UDP Packet Structure

Table A1.1 shows the structure of the UDP packet produced by our LiDAR sensor [20]. Note that LSB is short for least significant byte and MSB is short for most significant byte.

TABLE A1.1: The UDP packet structure

Bytes	**Description**
1-4	4 bytes timestamps (seconds)
5-8	4 bytes timestamps microseconds
9-12	4 bytes number of octets of packet saved in file
13-16	4 bytes actual length of packet
Ethernet frame	
17-22	Ethernet Destination address

Continued on next page

Table A1.1 – continued from previous page

Bytes	Description
23-28	Ethernet Source address
29-30	Ethernet message type (IP)
31-34	Header + Field Service + Total Length
35,36	ID
IP packet	
37	Flag
38	Offset
39	TTL
40	IP Protocol ID
41-42	IP Checksum
43-46	IP Source address
47-50	IP Destination address
UDP packet	
51-52	MSB, LSB Source port number
53-54	MSB, LSB Destination port number
55-56	Length
57-58	UDP Checksum
Shot 1	
59-60	Laser Block ID (0-31)
61-62	Angle
63-64	Distance (Laser 0)

Continued on next page

Table A1.1 – continued from previous page

Bytes	Description
65	Intensity (Laser 0)
66-67	Distance (Laser 1)
68	Intensity (Laser 1)
69-70	Distance (Laser 2)
71	Intensity (Laser 2)
⋮	⋮
156-157	Distance (Laser 31)
158	Intensity (Laser 31)
Shot 2-11	
⋮	⋮
Shot 12	
1159-1160	Laser Block ID (0-31)
1161-1162	Angle
1163-1164	Distance (Laser 0)
1165	Intensity (Laser 0)
1166-1167	Distance (Laser 1)
1168	Intensity (Laser 1)
1169-1170	Distance (Laser 2)
11 71	Intensity (Laser 2)
⋮	⋮
1256-1257	Distance (Laser 31)

Continued on next page

Table A1.1 – continued from previous page

Bytes	Description
1258	Intensity (Laser 31)
Total of (12 shots) × (32 lasers)	
1259-1260	Ethernet status
1261-1264	Ethernet checksum

Bibliography

[1] "Preliminary Statement of Policy Concerning Automated Vehicles", National Highway Traffic Safety Administration, 2013. [Online]. Available: https://www.nhtsa.gov/staticfiles/rulemaking/pdf/Automated-Vehicles-Policy.pdf.

[2] "2017 Cadillac ATS Product Information", Ford, 2016. [Online]. Available: http://media.cadillac.com/media/us/en/cadillac/vehicles/ats/2017.html.

[3] "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey", National Highway Traffic Safety Administration, 2015. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/View-Publication/812115.

[4] "Blind Spot Information System with Cross Traffic Alert ", 2017. [Online]. Available: https://owner.ford.com/how-tos/vehicle-features/safety/blind-spot-information-system-with-cross-traffic-alert.html.

[5] "Google Self-Driving Car Project Monthly Report", Google, 2016. [Online]. Available: https://www.google.com/selfdrivingcar/reports/report-0916.

[6] R. Wallace, and G. Silberg, "Self-Driving Cars: The Next Revolution, "

Connected Vehicle Technology, 2012.

[7] Z. Kalal, K. Mikolajczyk, and J. Matas,“Tracking-Learning-Detection, ” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012, vol. 34, no. 7, pp. 1409-1422.

[8] M. Malik, and S. Majumder,“An Integrated Computer Vision Based Approach For Driving Assistance To Enhance Visibility In All Weather Condition, ” *International and National Conference on Machines and Mechanisms*, 2013, Roorkee, India.

[9] G. Soysal, and M. Efe,“Performance Comparison of Tracking Algorithms For A Ground Based Radar, ” *The IEEE Seminar on Target Tracking: Algorithms and Applications* , 2006, pp. 1-16, Tandogan, Ankara.

[10] J. Besada, J. Garcia, G. De Miguel, A. Berlanga, J. Molina, and J. Casar,“Design of IMM filter for radar tracking using evolution strategies, ” *IEEE Transactions on Aerospace and Electronic Systems*, 2005, vol. 41, pp. 1109-1122.

[11] Z. Ding, and H. Leung,“Evaluation of two IMM-based Algorithms in Real Radar Tracking Environments, ” *Canadian Conference on Electrical and Computer Engineering* , 2008, pp. 1569-1574, Niagara Falls, ON, Canada.

[12] K. Li, X. Chen, and G. Zhou,“Maneuvering Target Tracking in Constraint Coordinates with Radar Measurements, ” *IEEE Radar Conference (RadarConf)*, 2016, Philadelphia, PA, USA.

[13] L. Meng, W. Grimm, and J. Donne,“Radar Detection Improvement by Integration of Multi-Object Tracking,” *Proceedings of the 5th International Conference on Information Fusion, IEEE Computer Society*, 2002, vol. 2, pp. 1249-1255.

[14] G.R. Widmann, M. K. Daniels, L. Hamilton, et al., “Comparison of Lidar-Based and Radar-Based Adaptive Cruise Control Systems, ” *SAE, Society of Automotive Engineers*, vol. 109, pp. 126–139, Detroit, Michigan, 2000

[15] D. Gohring, M. Wang, M. Schnurmacher, and T. Ganjineh, “Radar/Lidar Sensor Fusion for Car-following on Highways,” *The 5th International Conference on Automation, Robotics and Applications*, pp. 407-412, Wellington, New Zealand , 2011

[16] S. Laible, Y. N. Khan, and K. Bohlmann, “3D LIDAR- and Camera-Based Terrain Classification Under Different Lighting Conditions, ” *Autonomous Mobile Systems*, 2012, pp. 21-29.

[17] T. Bucher, C. Curio, and J. Edelbrunner, “ Image Processing and Behaviour Planning for Intelligent Vehicles, ” *IEEE Transactions on Industrial Electronics*, 2003, vol. 50, pp. 62-75.

[18] J. Choi, S. Ulbrich, B. Lichte, and M. Maurer, “Multi-Target Tracking using a 3D-Lidar Sensor for Autonomous Vehicles, ” *Proceedings of the 16th International IEEE Annual Conference on Intelligent Transportation Systems* , 2013, The Hague, The Netherlands.

[19] Q. Zhu, L. Chen, Q. Li, M. Li, A. Nuchter, and J. Wang, “3D LIDAR Point Cloud based Intersection Recognition for Autonomous Driving , ” *IEEE Intelligent Vehicles Symposium* , 2012, pp. 456-461, Alcalá de Henares, Spain.

[20] “Velodyne HDL-32E User’s Manual”, Velodyne, 2016. [Online]. Available: http://velodynelidar.com/lidar/products/manual/63-9113%20HDL-32E%20manual-Rev%20G.pdf

[21] S. Nobili, S. Dominguez, G G, P. Martinet, “16 Channels Velodyne Versus Planar LiDARs Based Perception System for Large Scale 2D-SLAM, " *The Workshop on Planning, Perception and Navigation for Intelligeng Vehicles*, 2015, pp. 6.

[22] J. Wei, J. Snider, J. Kim, et al., “Towards a Viable Autonomous Driving Research Platform,” *IEEE Intelligent Vehicles Symposium, Proceedings*, 2013, pp. 763-770.

[23] F. Moosmann, O, Pink, and C, Stiller, “Segmentation of 3D Lidar Data in non-flat Urban Environments using a Local Convexity Criterion,” *IEEE Intelligent Vehicles Symposium, Proceedings*, 2009, pp. 215-220.

[24] Z. Luo, S. Habibi, and M. Mohrenschildt, “LiDAR Based Real Time Multiple Vehicle Detection and Tracking,” *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2016, vol. 10, no. 6.

[25] L. Huang, and M. Barth, “Tightly-coupled Lidar and Computer Vision

Integration for Vehicle Detection," *IEEE Intelligent Vehicles Symposium*, 2009, pp. 604-609.

[26] S. Kammel, J. Ziegler, B. Pitzer, and M. Werling, "Team AnnieWAY's Autonomous System for the DARPA Urban Challenge 2007," *Internatinal Journal of Field Robotics Research*, 2008.

[27] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, and S. Thrun, "Junior: The Stanford Entry in the Urban Challenge," *Journal of Field Robotics*, 2008, vol. 8, pp. 69-597.

[28] S. Kammel, and B, Pitzer, "Lidar Based Lane Marker Detection and Mapping," *IEEE Intelligent Vehicles Symposium*, 2008, pp. 1137-1142.

[29] M. Himmelsbach, F. Hundelshausen, and H. Wuensche, "Fast Segmentation of 3D Point Clouds for Ground Vehicles," *IEEE Intelligent Vehicles Symposium*, 2010, pp. 560-565.

[30] C. Guo, W. Sato, and L. Han, "Graph Based 2D Road Representation of 3D Point Clouds for Intelligent Vehicles," *IEEE Intelligent Vehicles Symposium*, 2011, pp. 715-721.

[31] B. Douillard, J. Underwood, and N. Kuntz, "On the Segmentation of 3D LIDAR Point Clouds," *IEEE International Conference on Robotics and Automation*, 2011, pp. 2798-2805.

[32] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng, "Discriminative Learning of Markov Random Fields for

Segmentation of 3D Scan Data," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 169-176.

[33] B. Li, T. Zhang, and T. Xia, " Vehicle Detection from 3D Lidar Using Fully Convolutional Network," Baidu Research – Institute for Deep Learning.

[34] M. A. Fischler, and R. C. Bolles,"Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, 1981, pp. 381-395.

[35] D. Bagnell, "Occupancy Mapping: An Introduction," 2016. [Online]. Available: http://www.cs.cmu.edu/ 16831-f14/notes/F14/16831-lecture06-agiri-dmcconac-kumarsha-nbhakta.pdf.

[36] B. S. Everitt, and A. Skrondal, "The Cambridge Dictionary of Statistics," Cambridge University Press, ISBN 0-521-81099-X.

[37] Z. Ghahramani, "An Introduction to Hidden Markov Models and Bayesian Networks," *International Journal of Pattern Recognition and Artificial Intelligence*, 2001, pp. 9-42.

[38] E. Ivanjko, and I. Petrovic, "Experimental Evaluation of Occupancy Grid Map Improvement by Sonar Data Corrections," *International Symposium on Mediterrean Conference on Control and Automation*, 2005.

[39] Y. Manuel, A. Olivier, and L. Christian, "Update Policy of Dense Maps: Efficient Algorithms and Sparse Representation," *Field and Service Robotics*, 2008, pp. 23-33.

[40] C. Shalizi, “Extending Linear Regression: Weighted Least Squares, Heteroskedasticity, Local Polynomial,” 2016. [Online]. Available: http://www.stat.cmu.edu/ cshalizi/350/lectures/18/lecture-18.pdf.

[41] D. Bellhouse, “Generalized Least Squares Iteratively Reweighted Least Squares,” 2016. [Online]. Available: http://www.stats.uwo.ca/faculty/bellhouse/Generalized%20Least%20Squares-DaeroKim.pdf.

[42] M. Li, Q. Li, “Real-time Road Detection in 3D Point Clouds using Four Directions Scan Line Gradient Criterion,” Velodyne, 2009. [Online]. Available: http://velodynelidar.com/lidar/hdlpressroom/pdf/Articles/.

[43] A. W. Moore, “An Intoductory Tutorial on Kd-trees,” *Technical Report No. 209, Computer Laboratory, University of Cambridge*, 1991.

[44] E. Nevala, “Introduction to Octrees,” *Game Programming*, 2014.

[45] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nuchter, “Comparison of Nearest-Neighbor-Search Strategies and Implementations for Efficient Shape Registration,” *Journal of Software Engineering for Robotics*, 2012, vol. 3, no. 1, pp. 2–12.

[46] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees,” *Autonomous Robots*, 2013, vol. 34, no. 3, pp. 189–206.

[47] J. L. Bentley, and J. H. Friedman, “Data Structures for Range Searching,”

ACM Computing Surveys, 1979, vol. 11, no. 4, pp. 397-409.

[48] P. J. Besl, and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992, vol. 14, no. 2, pp. 239–256.

[49] J. Shen, J. Liu, R. Zhao, and X. Lin, "A Kd-tree-based Outlier Detection Method for Airborne LiDAR Point Clouds," *IEEE International Symposium on Image and Data Fusion* , 2011, pp. 1-4.

[50] S. Ying, G. Xu, C. Li, and Z. Mao, "Point Cluster Analysis Using a 3D Voronoi Diagram with Applications in Point Cloud Segmentation," *International Journal of Geo-Information* , 2015, pp. 1480-1499.

[51] Y. Yu, J. Li, H. Guan, F. Jia, and C. Wang, "Three-Dimensional Object Matching in Mobile Laser Scanning Point Clouds," *IEEE Geoscience and Remote Sensing Letters* , 2014, pp. 492-496.

[52] H. Frigui and R. Krishnapuram, "Clustering by Competitive Agglomeration," *Pattern Recognition Journal*, 1997, pp. 1109-1119.

[53] D. Boley, "Principal Direction Divisive Partitioning," *Data Mining Knowledge Discovery*, 1998, pp. 325-344.

[54] F. Pauling, M. Bosse, and R. Zlot, "Automatic Segmentation of 3D Laser Point Clouds by Ellipsoidal Region Growing," *Australasian Conference on Robotics and Automation*, 2009, pp. 1-10.

[55] K. Klasing, D. Wollherr, and M. Buss, “A Clustering Method for Efficient Segmentation of 3D Laser Data,” *IEEE International Conference on Robotics and Automation*, 2008, pp. 4043-4048.

[56] D. Dworak, “3D Points Cloud Reduction Using Modified K-D Tree Method,” 2009.

[57] M. Ester, H. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” *International Conference on Knowledge Discovery and Data Mining*, 1996.

[58] D. Davies, and D. W. Bouldin, “A Cluster Separation Measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979.

[59] J.C. Dunn, “Well Separated Clusters and Optimal Fuzzy Partitions,” *Cybernetics*, 1974, pp. 95-104.

[60] C. Cortes, and V. Vapnik, “Support-Vector Networks,” *Machine Learning*, 1995, pp.273-297.

[61] C. Zhang, X. Chen, and W. B. Chen, “A PCA-Based Vehicle Classification Framework,” *IEEE International Conference on Data Engineering*, 2006, pp. 17-17.

[62] Z. Qian, H. Shi , J. Yang and L. Duan, “Video-Based Multiclass Vehicle Detection and Tracking,” *IJCSI International Journal of Computer Science Issues*, 2013, vol. 10, no. 3.

[63] S. Zehang, G. Bebis, and R. Miller, “On-Road Vehicle Detectionusing Evolutionary Gabor Filter Optimization,” *IEEE Transactions on Intelligent Transportation Systems*, 2005, pp.125-137.

[64] A. Krizhevsky, I. Sutskever, G. E. Hinton, “Imagenet Classification with Deep Convolutional Neural Networks,” *In Advances in neural information processing systems*, 2012, pp.1097-1105.

[65] A. Börcs, B. Nagy, and C. Benedek, “Instant Object Detection in Lidar Point Clouds,” *IEEE Geoscience and Remote Sensing Letters*, 2017, pp.1-5.

[66] M. Kirby and L. Sirovich, “Application of the Karhunen-Lokve Procedure for the Characterization of Human Faces,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990, pp.103–108.

[67] S. Gupte, O. Masoud, R. F. K. Martin, and N. P. Papanikolopoulos, “Detection and Classification of Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, 2002, vol. 3, no. 1, pp. 37-47.

[68] I.T. Jolliffe,“Principal Component Analysis, second edition”, *Springer*, 2002.

[69] R. Blomley, M. Weinmanna, J. Leitloffa, and B. Jutzi, “Shape Distribution Features for Point Cloud Analysis - A Geometric Histogram Approach on Multiple Scales,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2014.

[70] J. F. Lalonde, N. Vandapel, D. Huber, and M. Hebert, “Natural Terrain

Classification Using Three-Dimensional Ladar Data for Ground Robot Mobility," *Journal of Field Robotics*, 2006, pp.839-861.

[71] N. Vandapel, D. F. Huber, A. Kapuria, and M. Hebert, "Natural Terrain Classification using 3-D Ladar Data, " *IEEE International Conference on Robotics and Automation*, 2004, vol. 5, pp. 5117-5122.

[72] V. Vapnik, "The Nature of Statistical Learning Theory," *Springer-Verlag*, 1995, ISBN 0-387-98780-0.

[73] L. Wang, "Support Vector Machines: Theory and Applications," *Springer*, 2005.

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton, " ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, 2012.

[75] M. Hossin, and M. N. Sulaiman, " A Review On Evaluation Metrics For Data Classification Evaluations ," *International Journal of Data Mining and Knowledge Management Process* , 2015, vol.5, no.2.

[76] R. E. Kalman "A New Approach to Linear Filtering and Prediction Problems ," *ASME*, Vol. 82, pp. 35–45,1960.

[77] M. S. Grewal and A. P. Andrews "Kalman Filtering: Theory and Practice Using MATLAB," *Wiley*, 2008, New York, USA.

[78] P.A. Boysen, and H. Zunker "Low Cost Sensor Hybridisation and Accuracy Estimation for Road Applications," *ESA Conference Navitec*, 2004.

[79] C. Boucher, A. Lahrech, and J.C. Noyer “Non-linear Filtering for Land Vehicle Navigation with GPS Outage,” *IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[80] S. Rezaei, and R. Sengupta “Kalman Filter Based Integration of DGPS and Vehicle Sensors for Localization,” *IEEE Transactions on Control Systems Technology* , 2007, vol. 15, no. 6.

[81] S. R. Habibi “The Smooth Variable Structure Filter,” *IEEE Invited Paper*, 2007, vol. 95, no. 5.

[82] V.I. Utkin “Sliding Mode Control Design Principles and Applications to Electric Drives,” *IEEE Transactions on Industrial Electronics*, 1993, vol. 40, no. 1.

[83] M. Attari, Z. Luo, and S. R. Habibi, “An SVSF-Based Generalized Robust Strategy for Target Tracking in Clutter,” *IEEE Transactions on Intelligent Transportation Systems*, 2016, vol. 17, no. 5.

[84] N. Hoyningen-Huene, and M. Beetz, “Robust Real-Time Multiple Target Tracking,” *Asian Conference on Computer Vision*, 2009, pp. 247-256.

[85] K. Oksuz and A. T. Cemgil, “The Comparison of the Performances of Global Nearest Neighbor and Probability Hypothesis Density Filter for Varying Clutter Rates,” *Signal Processing and Communication Application Conference*, 2016, pp. 677–680.

[86] C. Rasmussen, and G.D. Hager, “Probabilistic Data Association Methods

for Tracking Complex Visual Objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , 2001, vol. 23, no. 6.

[87] Z. Ding, and L. Hong , "Bias Phenomenon And Compensation for PDA/JPDA Algorithms" *Mathematical and Computer Modelling* , 1998, vol. 27, no. 12.

[88] R. Kalman"A New Approach to Linear Filtering and Prediction Problems," *Transaction of the ASME, Journal of Basic Engineering.* 1982, pp. 35.

[89] S. A. Gadsden, "Smooth Variable Structure Filter: Theory and Application," *McMaster University*, 2011.

[90] M. Attari, S. A. Gadsden, and S. R. Habibi, "Target Tracking Formulation of the SVSF with Data Association Techniques," *IEEE Transactions on Aerospace and Electronic Systems*, 2014.

[91] D. G. Luenberger, "Introduction to Dynamic Systems," *New York: Wiley*, 1979.

[92] H. W. Kuhn, "The Hungarian Method for the Assignment and Transportation Problems," *Naval Research Logistics Quarterly*, 1995, pp. 83-97.

[93] Y. Bar-Shalom, and E. Tse, "Tracking in Cluttered Environment with Probabilistic Data Association," *Automatica*, 1975, vol. 11, no. 5.

[94] Y. Bar-Shalom, F. Daum, and J. Huang, "The Probabilistic Data Association Filter, Estimation in the Presence of Measurement Origin

Uncertainty," *IEEE Control Systems*, 2009, vol. 29, no. 6, pp. 82-100.

[95] D. P. Bertsekas, "Introduction to Probability," Sections 1.3-1.4.

[96] T. Kirubarajan, and Y. Bar-Shalom, "Target Tracking Using Probabilistic Data Association Based Techniques with Applications to Sonar, Radar, and EO Sensors," *CRC Press*, 2011, ISBN 978-0-8493-2379-9.

[97] Y. Bar-Shalom, and X. R. Li, "Multitarget-Multisensor Tracking: Principles and Techniques," *Storrs, CT: YBS Publishing*, 1995.

[98] D. A. Brannan, M. Esplen, J. J. Gray, "Geometry," Cambridge University Press, ISBN 0-521-59787-0.

[99] M. Attari, S. A. Gadsden, and S. Habibi, "Target Tracking Formulation of the SVSF as a Probabilistic Data Association Algorithm," *American Control Conference* , 2013.

[100] H. A. Blom, and Y. Bar-Shalom, "The Interacting Multiple Model Algorithm for Systems with Markovian Switching Coefficients," *IEEE Transaction on Autonomous Control*, 1988, pp. 780-783.

[101] C. Esposito, A. Castiglione, F. Palmieri, M. Ficco, and K. R. Choo, "A Publish/Subscribe Protocol for Event-Driven Communications in the Internet of Things," *IEEE International Conference on Pervasive Intelligence and Computing*, 2016, Auckland, New Zealand.

[102] "Profile to Improve Performance", MathWorks, 2017. [Online]. Available:

https://www.mathworks.com/help/matlab/matlab-prog/profiling-for-improving-performance.html

[103] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, M. Cifrek, "A Brief Introduction to OpenCV," *IEEE Proceedings of the 35th International Convention*, 2012, Opatija, Croatia.