LIDAR BASED OBJECT DETECTION AND TRACKING IN STATIONARY APPLICATIONS

A Thesis

by

AMIR A. DARWESH

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Dr. Srikanth Saripalli
Co-Chair of Committee,	Dr. James E. Hubbard, Jr.
Committee Members,	
	Dr. Dylan Shell
Head of Department,	Dr. Bryan Rasmussen

December 2021

Major Subject: Mechanical Engineering

ABSTRACT

This thesis investigates dense Light Detection and Ranging (LiDAR) sensors as a method for object detection and tracking in stationary infrastructure-like applications. A literature review of existing works is conducted, with discussion and comparisons for other sensing technologies. Additional discussions are made for geometric feature-based methods and end-to-end learning methods for object detection from pointcloud data. Subsequently, theoretical pointcloud spacing models for multi-beam 360° LiDAR sensors are developed, with analysis on placement strategies and LiDAR configurations. The thesis continues with an implementation of a geometric feature based object detection method, primarily for vehicles. Several algorithm designs are presented for pointcloud background removal, clustering, orientation detection, tracking, and filtering. Detection and tracking metrics are then established to observe the system's performance on both experimental and simulation datasets. Two datasets collected with a Velodnye VLP-16 sensor on both a highway and urban road segment are utilized for experimentation, while scenarios of light traffic and stop-and-go traffic on a highway are developed in the CARLA simulator to further validate tracking performance.

DEDICATION

To my parents, brother, sister, and Shreya

ACKNOWLEDGMENTS

I would like to thank Dr. Srikanth Saripalli for his excellent mentorship and guidance over the past four years when I first joined the Unmanned Systems Lab as an undergraduate student.

Additionally, I would like to thank my committee members, Dr. James E Hubbard Jr., and Dr. Dylan Shell for their feedback and thoughts that helped me progress through my Master's program.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Srikanth Saripalli (advisor) and Professor James E. Hubbard Jr. (co-advisor) of the J. Mike Walker '66 Department of Mechanical Engineering and Professor Dylan Shell of the Department of Computer Science & Engineering.

Results from two of the experiments in Chapter 5 were obtained through previous work of the student, and was published in a 2021 conference article in the Intelligent Transportation Systems Conference (ITSC).

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported through RA positions held in the Unmanned Systems Lab led by Dr. Srikanth Saripalli.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction

The objective of multi object detection and tracking (MODT) has been studied for a variety of applications and motivations, such as military defense and surveillance purposes, robotics applications including autonomous driving, infrastructure use for traffic monitoring, asteroid detection and tracking, or even in sports tournaments such as ball tracking in soccer and football. Historically, the sensing technologies enabling object detection and tracking have primarily been computer vision or RADAR based, upon which vision based technologies are most currently used today [4]. Although these sensing technologies have seen numerous successes in their applications, no individual sensing modality is without weakness. Hence, it warrants that new sensing technologies be thoroughly investigated, as their properties may be more advantageous in certain domains or conditions.

1.1.1 LiDAR Sensors

Early forms of Light Detection and Ranging (LiDAR) sensors were introduced over 60 years ago, with early detectors being more akin to today's common household laser based measuring tool. Their operation relies upon Time of Flight (ToF) principles, where the amount time is measured between the instance when a pulse of light is shined towards an object, and the instance the light's reflection reaches the original source. By assuming minute atmospheric interference, the distance can be calculated by multiplying the time measured by the speed of light, and halving to retrieve the one-way distance:

$$d = \frac{ct}{2} \tag{1.1}$$

There are many different types of LiDAR sensors, and are often separated by both the spatial-dimensionality they measure in, and pointcloud density that they produced, as illustrated by Figure 1.1. One-dimensional LiDARs can be envisioned as a simple laser distance measurement tool, commonly used in home-improvement projects. One dimensional LiDARs capture a single

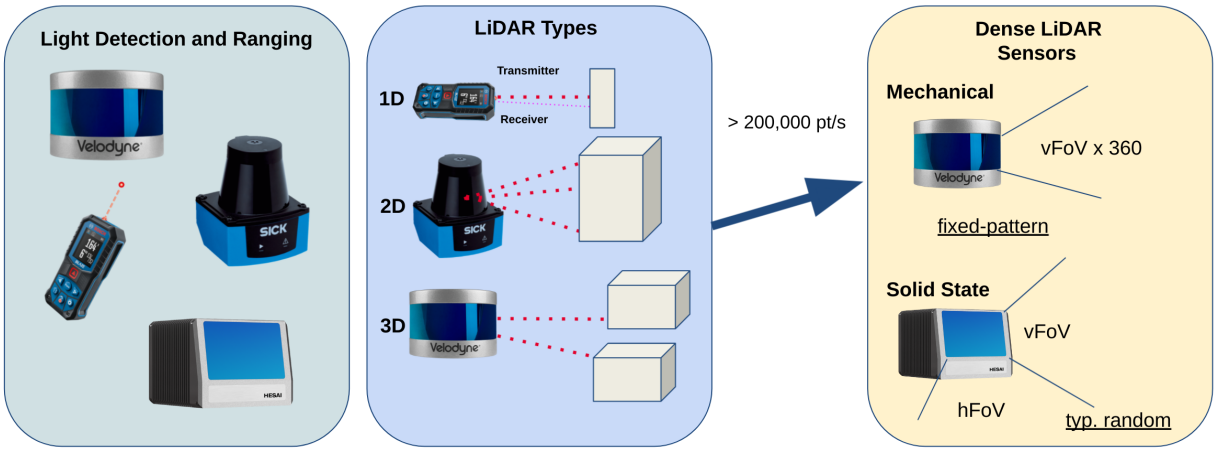


Figure 1.1: Different types of LiDAR, separated by dimensionality, and density

point in-space, and are typically only used for range-finding. Two-dimensional LiDARs can be envisioned as a one-dimensional LiDAR rotating at a controlled and precise angular velocity. By precisely controlling the yaw angle of the LiDAR through precise motion encoders, distance measurements can be taken in a single plane and represented by Polar coordinates (r, ψ), or Cartesian coordinates (x, y). Similarly, 3D spinning LiDARs can be envisioned as multiple 2D LiDARs stacked together vertically at set angles, where each angle represents a scanning channel. Distance measurements can be represented as spherical coordinates $(r, \theta, \psi,)$, or as Cartesian coordinates (x, y, z). Due to their precise rotation and firing sequences, spinning or mechanical LiDAR sensors have a great degree of determinism in their scanning pattern. In this context, scanning pattern refers to each laser measurement taken correlating to a particular channel and yaw rotation. Within the past decade and a half, the $360°$ horizontal scanning version of the sensing technology has seen rapid industry development. $360°$ LiDAR sensors have multiple beams that scan 360 degrees in a panoramic fashion, typically with 16, 32, 64 or 128 beams of lasers. There are also solid-state LiDARs that exist today, which do not spin mechanically but instead use a microscopic electro-mechanical sensor array that produces a three-dimensional point cloud. Solid-state LiDARs however are not deterministic in their scanning pattern, and are typically random varying with the sampling time.

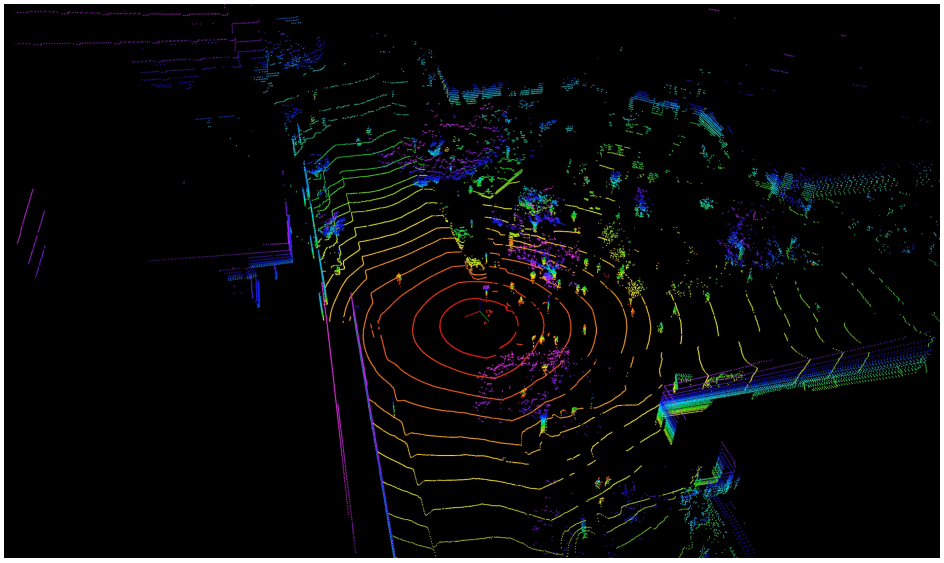


Figure 1.2: Example of LiDAR data taken near Ireland St., College Station TX with a Velodyne VLP-32c sensor. Here each color represents the channel of the LIDAR sensor.

When referred to today, LiDAR today is often envisioned as the dense version of the sensor, where three dimensional clouds of objects and background within a vertical and horizontal field of view. An example of LiDAR data is visualized in Figure 1.2. Companies manufacturing LiDAR sensors have seen recent significant uptake in industry demand, as the sensing technology has had numerous applications in robotics and other innovative fields. In fact, due to much speculation of their widespread adoption, retail and professional investors have taken a hold to market watch over established and new LiDAR manufacturers that are now being traded publicly [5].

1.1.2 Applications

Dense LiDAR sensors have seen many uses in different fields, ranging from terrestrial, air-borne, infrastructure, and robotics applications [6]. More narrowly, 360° and other similar dense scanning LiDAR sensors have been vital sensing components for robotics and autonomous driving companies. Their ability to provide dense clouds of points has seen utilization for robust obstacle detection, Simultaneous Localization and Mapping (SLAM), object detection and tracking, road marking detection, signage detection, and other use-cases. Outside robotics, other applications for dense point-cloud LiDAR sensors include uses for Geographical Information Systems (GIS),

Table 1.1: Comparison of different sensors

Sensor Type	Illumination Req	Classification Ability	Range	Resolution	Direct 3D Space Representation
RADAR	Independent	Low	High	Low	Yes
LIDAR	Independent	Medium	Medium	Medium	Yes
Monocular Cameras	Dependent	High	Mixed	High	No
Stereoscopic Cameras	Dependent	High	Short	High	Yes
Thermal Cameras	Independent	High	Medium	Medium	No

sensing in infrastructure based "smart-cities", and uses for security systems.

1.1.2.1 Dynamic vs. Stationary Applications

One method to sub-categorize applications of dense point-cloud LiDAR sensors is to group by dynamic and stationary applications. Attaching the LiDAR to a moving structure, such as a vehicle or robot platform, offers the broadest range of utility for the LiDAR sensor when compared to a fixed mounting location. Applications where LiDARs are fixed in location relative to their scene are narrower; for example, they may be used for developing 3D models of a building structure, geology feature, or other similar items. Similarly to dynamic applications, fixed location LiDARs can also be used for object detection and tracking and have seen prior research in infrastructure based traffic monitoring.

1.1.3 Other Sensors

It is also important to compare LiDAR sensors against other sensing technologies. The next few subsections introduce other sensing technologies commonly used, with Table 1.1 summarizing.

1.1.3.1 Cameras

Color monocular cameras, usually referred to as just cameras, are passive sensors that measure in the visible light spectrum (350–750 μm) and return a two-dimensional image with color information. Several models of cameras are available commercially off the shelf, and vary in resolution, shutter type, shutter speed, and form-factor. Due to the high resolution often provided by

cameras today, they prove to be excellent in use of object detection and classification under normal operating conditions. However, they are lighting dependent, and poorly illuminated scenes make for difficult recognition of objects. Further, it is challenging to translate a 2D pixel value to a point into 3D space unless the intrinsic and extrinsics of the sensor are estimated accurately.

1.1.3.2 Stereoscopic Cameras

Stereoscopic cameras, or stereo cameras, utilize two separate in-line imaging sensors separated by a known distance to capture two perspectives of a scene. Because the separation distance is known, a three dimensional view of the scene can be reconstructed and is usually provided with the sensor output. However, commercially available stereo cameras are typically limited by the depth they can measure, and are also subject to the same illumination weaknesses as monocular cameras.

1.1.3.3 Thermal Cameras

Thermal cameras are much like monocular cameras, but instead passively measure in the infrared-spectrum and therefore are not subject to illumination conditions like cameras. Objects that emit heat also emit infrared radiation, which make thermal cameras particularly useful for detecting living objects. However, the 2D pixel data is still difficult to translate to a 3D point in space. Additionally, due to requiring a larger sensor array to measure larger wavelengths compared to normal cameras, thermal cameras are usually lower in sensor resolution.

1.1.3.4 RADAR Sensors

Radio Detection and Ranging (RADAR) sensors have been been around for over 80 years, and have had numerous applications including military, space, robotics, meteorology, and several others. Their principle operates similarly to active LiDAR sensors, but instead use much larger radio-waves (0.8-10 cm). An emitter will distribute pulsed radio waves where a receiver captures any reflected return signals revealing an objects position, speed (due to Doppler shift), and bearing from the sensor. Due to the longer wavelengths emitted, their range can be great distances depending on the power and antenna array. However, compared to both camera and LiDAR sensors,

RADARS are much lower in resolution making classification of objects challenging.

1.1.4 Sensor Fusion

As seen, no individual sensing modality is without weakness. One method to reduce perception weaknesses in a system is by utilizing multiple sensing modalities that can complement each-other. Several concepts exist for leveraging multiple senor's outputs to generate robust detections and tracks. In most sensor fusion methods, precise extrinsic calibrations or parameters that define the rigid transformations between sensor coordinate frames are required to transform sensor or object data into a single frame of reference. Two popular concepts for sensor fusion are briefly discussed.

1.1.4.1 Early Fusion

In early sensor fusion, raw sensor data is usually transformed into a single reference frame and then combined into a higher order state representation. Detection and classification algorithms are then subsequently computed on the combined sensor data array.

1.1.4.2 Late Fusion

In late fusion methods, individual detection and classification algorithms are developed for each sensing modality, where detections are later fused and or filtered into a single lower order representation.

Although their use-cases are many, LiDARs are interesting to research from an object detection and tracking perspective. They are higher in resolution compared to radio based ranging technologies (RADAR), and are direct sources for 3D measurements that is not possible with monocular vision cameras.

1.2 Research Scope

Although LiDARs can be used for many other methods, this thesis investigates LiDAR as a sensing technology for object detection and tracking, primarily in fixed infrastructure-like applications. A summary of the objectives and contributions for this work is presented in the following subsection.

1.2.1 Objectives and Contributions

The objectives of this work are mostly expository, where methods for LiDAR based object detection and tracking are both investigated and implemented. Objectives include:

- A literature review of previous works for object detection and tracking using LiDAR sensors;
- A review of current generation dense point-cloud LiDAR sensors;
- Analysis and discussion over LiDAR resolution, frequency, and range as it relates to vehicle detection;
- Implementation of object detection and tracking algorithms in both simulation and experimentation;
- Defining object detection and tracking metrics to provide meaningful analysis;
- Comparing results obtained via simulation against experimentation to validate algorithm performance.

1.3 Literature Review and Related Works

Significant interest of dense point-cloud LiDAR technology has produced a number of prior works for both dynamic and stationary applications. Object detection and tracking remains an area of active research, with recent interest in develop object detectors using neural networks. Literature review for LiDAR based MODT has been divided into three categories: (1) Feature based detection methods, (2) Learning based detection methods, and (3) Multi Object Tracking (MOT) algorithms.

1.3.1 Feature based detection methods

Feature or model based object detection methods utilize programmatic algorithms to remove background points, and cluster points into representative object detections. Variations in these methods are primarily in how background points are removed, how points are clustered, and how meta descriptors such as the centroid location and velocity are estimated.

Research in 2019 from Sualeh and Kim developed a LiDAR based MODT system primarily for vehicle detection and tracking using multiple LiDARs equipped on a moving research vehicle. Their methodology utilizes a ground plane removal technique using a grid based approach, and a 3D cell based method for pointcloud clustering [4]. A minimum area and L-shape fitting algorithm is then used to estimate detection position and orientation. Similar approaches were taken by [7] for LiDAR based vehicle detection, though a hierarchical 3D grid model is utilized for both clustering and separation of foreground points. Convex hulls are then fitted and utilized for rectangle fitting with orientation.

Moving to works in stationary applications, [8] details methods used for a road side LiDAR application for vehicle tracking and speed estimation. Algorithms for background removal are much simpler in stationary applications, as the background scene does not change little compared to the dynamic case. Their methodology assumes an impenetrable environment, and therefore the furthest recorded scanning distances over a configurable period of time are classified as background points. For point cloud clustering, a three parameter Euclidean Clustering algorithm is used on the 3D point-cloud, where then clusters undergo binary classification as vehicle or non-vehicles. Their methodology tests three different classifying methods, including Support Vector Machine (SVM) based classifiers, Random Forest classifiers, and rule-base classifiers. Results showed that differences in the three classifiers performance were nearly indistinguishable, with the authors favoring the rule-based classification method to avoid false-negative classifications. A 2019 dissertation by J. Zhao [9] also investigated infrastructure LiDAR applications for both vehicle and pedestrian detection and tracking. Similarly, their methodology for background filtering assumes a impenetrable in environment, and generates a table of maximally recorded distances for each beam and scan angle of the LiDAR. For clustering, they utilize a modified version of the Density Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm that has adaptive parameters for the minimum number of cluster points and searching radiii. For classification between vehicle and pedestrian, a backpropagation artificial neural network (BP-ANN) is used with three features: (1) total number of cluster points, 2D distance from the sensor, and (3) the direction distribution of the

cluster points.

1.3.2 Learning based detection methods

Learning based methods instead rely upon by training a neural network with a set of annotated LiDAR data. Typically, data is not pre-processed like feature based methods to remove background points, which make learning based methods more advantageous in this regard. However, learning based methods add computational complexity, often requiring significant hardware investment to run in real-time between sensor updates.

Typically, in object detection for RGB imagery, convolutional neural networks (CNNs) are trained and then used for inference to produce object detections and classifications. Because CNN architectures are developed for 2 dimensional images (typically with 3 color channels), they cannot directly be used on 3D pointcloud data. However, a work-around presented in [10] instead projects the 3 dimensional LiDAR data into a depth image such that the input to the CNN remains a 2D image (with depth encoded as color). Subsequently, by using annotated LiDAR frames for vehicles and other classes, the CNN can be trained to detect objects.

More generally, recent works have been investigating semantic segmentation of LiDAR data, where every LiDAR point is classified. The number of publications detailing new research over semantic LiDAR segmentation is increasing significantly, often presented by a moniker for the network name - e.g. Rangenet++ [11], PointSeg [12], and Polarnet [13]. These works primarily vary on the type of image projection model used (e.g. spherical, cylindrical, Bird's Eye View and others), as well as over the CNN architecture.

Although the proposed research scope does not include new research on Neural network object detection based methods, they are powerful methods, and should also be considered as a valid method for MODT.

1.3.3 Multi-Object Tracking methods

Multi-object tracking (MOT) refers to the process in which new object detections are reconciled into tracked objects. MOT is usually not dependent on what method was utilized to obtain the

detection from a sensor, as it instead tracks heuristic features, commonly position and velocity. Typically, multi-object tracking follows a conventional process where new detections are matched or associated with existing tracks, and subsequently filtered to reduce noise. During times where no detections are received, the filter mechanism can instead utilize a motion model for predicting the the next states of the tracked object. Existing works for LiDAR based have primarily been similar in methodology for the detection association and filtering processes; methods have utilized the Join Probabilistic Data Association Filter (JPDAF) coupled with the popular Kalman Filter [4], [8]. Other methodologies still utilize Kalman filters, but instead use distance based cost functions with new detections and prior tracks to determine the associations (e.g. the Hungarian Munkres Algorithm) [9], [14]. Another method for data association is through visual association rather than cost based or probabilistic matching from position. [8] also included a tracking refinement module where detections from LiDARs in subsequent frames were score matched by comparing 2D Bird's Eye View image projections to correlate the same object detections between frames. Similar visual score matching concepts have also been used in computer vision based MOT, such as the SORT method [15]. Visual matching methods for data association are advantageous in that a more direct sensor reading is matched rather than computed heuristics such as centroid position, which can often be noisy due to partial readings.

2. THEORY AND ALGORITHM DESIGN

This chapter focuses on the theoretical aspects of LiDAR sensors and design of algorithms for multi object detection and tracking in a stationary LiDAR application.

2.1 System Requirements

At a high-level, an object detection and tracking system can seek to accomplish the following:

1. Detect 2D or 3D positions and bounding boxes from objects of interest in LiDAR data;
2. Detect an object's velocity;
3. Classify detections (e.g. noise, pedestrians, vehicles);
4. Filter, and track individual objects state-history;
5. Maintain creation and deletion of tracks as they enter and leave the scene.

More narrowly, a process can be tailored specifically for LiDAR sensors, as presented by Figure 2.1.

2.2 Fixed Pattern LiDAR Theory

In Section 1.1.1, the concept of a fixed-pattern scanning LiDAR was introduced. This section intends to cover theoretical point-cloud distribution models for fixed-pattern LiDAR sensors. Figure 2.2 illustrates how points in the XY plane separate with the azimuth (horizontal) angle taken

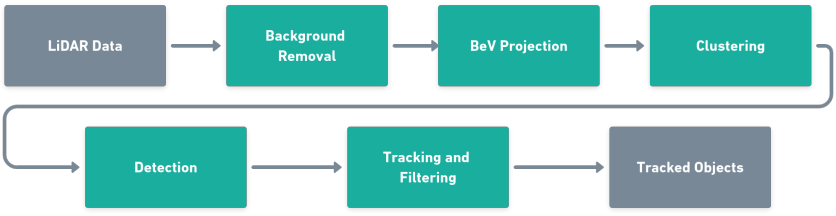


Figure 2.1: Flow-Diagram of a geometric-feature based method of object detection and tracking

against a flat wall at a distance D_p away with a 2D LiDAR scanner. As the the total azimuth angle $n\varphi$ increases counter-clockwise from the x-axis, so does the horizontal spacing A_n between points. The relationship between $A_n, \varphi,$ and D_p can be governed by the following trigonometric relation:

$$A_n = D_p(\tan n\phi - \tan(\phi(n-1)) \tag{2.1}$$

where n ranges from $1 \rightarrow \frac{\pi}{\phi}$, with φ as the angular resolution of the LiDAR. Note that the azimuth / horizontal resolution is also usually proportional to the angular rotation rate ω of the LiDAR due to having a fixed total number of points scanned per second:

$$\varphi \propto \omega \tag{2.2}$$

This follows that as the LiDAR's rotation rate increases, the spacing distances A_n also subsequently increase.

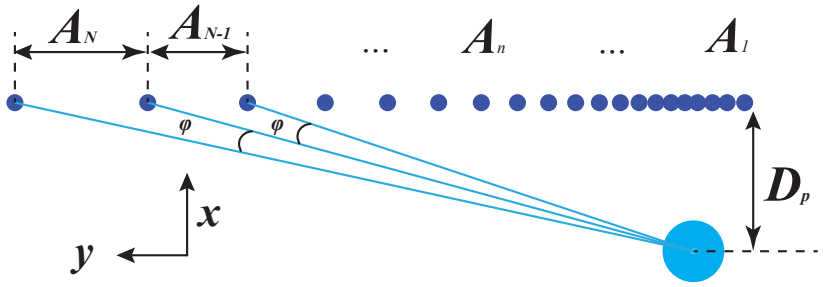


Figure 2.2: Horizontal spacing in XY plane between points for a 2D scan of a flat wall with the LiDAR sensor positioned D_p away. n ranges from $N = 1$ to $N = \frac{\pi}{\varphi}$ [1]. ©2021 IEEE

With a 3-dimensional LiDAR sensor, point-cloud spacing analysis can be conducted over multiple beams of the LiDAR in the XZ plane, as illustrated in Figure 2.3 for three channels of the LiDAR. In this example, the LiDAR is positioned away from some region of interest both vertically and horizontally at D_v, D_h, respectively. Directly underneath the LiDAR lies a conically shaped blind-spot that varies primarily based on the mounting height and the sensor's vertical field

of view. The blind-spot can be characterized as a function of horizontal set-back distance to the region of interest D_h, and Γ, the lower bound of the vertical field of view:

$$\delta_{\text{min}} = D_v - D_h \tan \Gamma \tag{2.3}$$

Point-cloud spacings (B_k) in the x direction at height δ_{min}, and vertical spacings (C_k) in the z

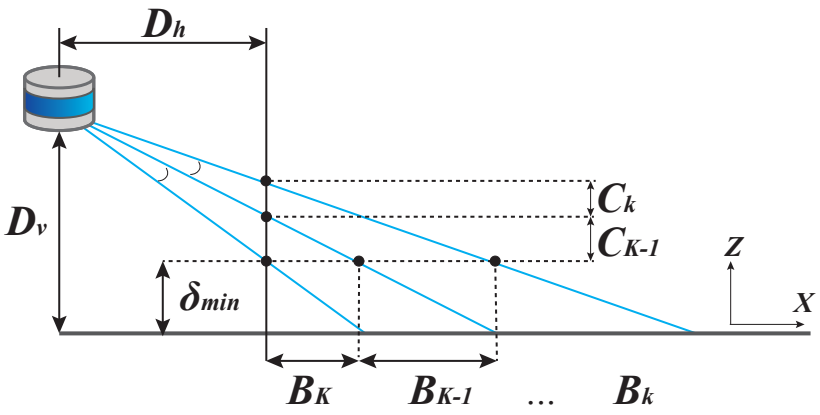


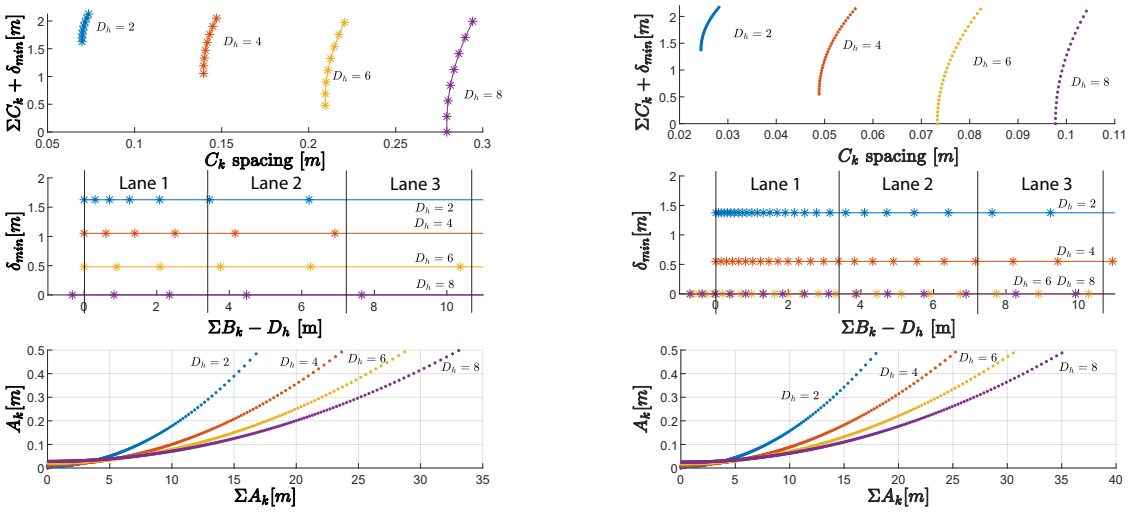
Figure 2.3: Horizontal/Vertical spacing between points in the ZX plane at δ_{min}. The mounting height D_v, and setback distance D_h from the road affect the spacing [1]. ©2021 IEEE

direction at set-back distance D_h are given by Equations (2.4) and (2.5), respectively. Although there are several other spacing values that can be computed, spacing values B_k and C_k are useful when conducting placement designs for a stationary LiDAR in infrastructure applications where vehicle detection is a priority. B_k can be though of representing the spacing between points on a vehicles roof, while C_k represents the vertical spacings along the side of a vehicle.

$$C_k = D_h(\tan k\gamma - \tan(\gamma(k-1)) \tag{2.4}$$

$$B_k = \frac{D_v - \delta_{\text{min}}}{\tan k\gamma - \tan(\gamma(k-1)} \tag{2.5}$$

Figures 2.4a and 2.4b illustrate variations on the the spacing value B_k, C_k, and A_k calculated



(a) Velodyne VLP-16, $D_v = 2.2m\ \omega = 10Hz$

(b) OS1-32, $D_v = 2.2m\ \omega = 10Hz$

Figure 2.4: A_k, B_k, C_k theoretical spacing results. $\Sigma C_k + \delta_{min}$ represents the distance from the ground, $\Sigma B_k - D_h$ represents the horizontal spacing along the lanes, while ΣA_k is the longitudinal spacing along the lane [1]. ©2021 IEEE

with two different LiDAR sensors at a fixed vertical height $D_v = 2.2m$ at different set-back distances D_h. The Velodyne VLP-16 LiDAR sensor is a 16-channel sensor with a $\pm 15°$ FoV, while the OS1-32 is a 32-beam LiDAR sensor and can be configured to scan below the horizon up to $-22.5°$. Note that only the channels that are lower than or equal to $0°$ are plotted for the VLP-16, and the calculated values for A_k are assumed for a LiDAR channel that has a vertical scanning angle of $0°$. Reviewing the top plots in Figure 2.4a shows that the spread of vertical spacings C_k increase as the set-back distance D_h increases, which follows intuition. However, as the set-back distance D_h increases, so does the spacing in the x direction, B_k, as seen by the middle plots. Also revealed in the middle plots is that as the set-back distance increases, the minimum height δ_{min} that describes the conical blind spot decreases.

With the spacing values having interdependencies on D_v, D_h, and ω, there are several competing metrics. The following would like to be accomplished:

- Choose a vertical mounting height D_v such that closest objects in-front of the LiDAR do not

completely occlude objects that may be behind;

- Choose an appropriate setback distance, such that the spread of vertical spacing's C_k is maximal;
- Choose an appropriate setback distance, such that the minimum height δ_{min} is kept at or below the mid-line of the smallest object to be detected;
- Choose an appropriate setback distance and rotation rate ω, such that the horizontal spacing A_k is minimized.

With these criteria listed, and from comparing the plots in Figures 2.4a and 2.4b, it can be seen that below-scanning LiDARs are more favorable due to the full number of beams being available for scanning objects below it's mounting height. LiDARs such as the VLP-16 and others that have a equal vertical field of view about its mid-plane must be instead mounted at shorter heights to utilize beams that scan above the mid-plane.

2.3 Background Filtering

Background filtering of the LiDAR point cloud aids in cluster extraction for vehicle detection. Generally, background points are readings from fixed objects in the environment, such as roads, buildings, and other infrastructure. Some non-stationary objects such as trees, bushes, and other vegetation (due to wind) can also be considered as background points, though it is more challenging to programmatically classify as such.

In feature based methods for vehicle detection, background points, such as readings from the ground plane, are desirable to be filtered out to easily segment vehicle clusters. There have been multiple methods developed to filter out background points in both dynamic and stationary applications of LiDAR, with the former having increased complexity.

2.3.1 Ground Plane Segmentation

Ground plane segmentation is the process in which ground readings are identified, usually to be removed. A common methodology for ground plane segmentation is performed via plane

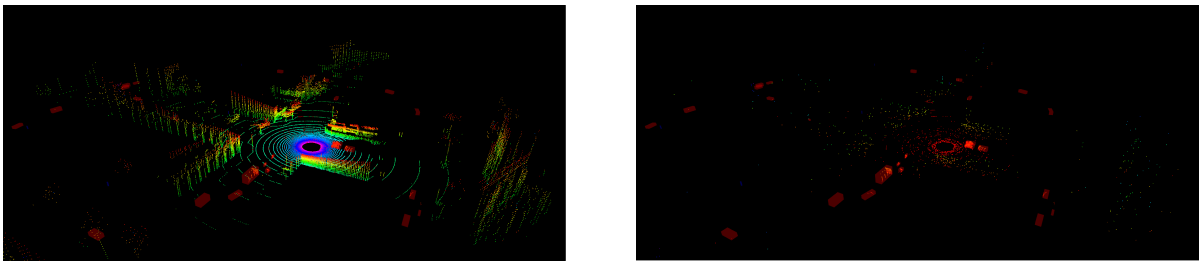
regression, where normal coefficients for the plane equation, shown below in Equation (2.6), are typically fitted via sample census methods [16].

$$ax + by + cz + d = 0 \tag{2.6}$$

Ground plane regression can be both performed in dynamic and stationary LiDAR applications; however, ground-plane removal does not remove other commonly found background points; for example, shoulder railings, signage, buildings and other objects are all points that may be desirable to remove prior to clustering. Hence, unless the detection area is constrained to a flat road surface without any additional scenery, additional or other filtering methods are required to remove background points that are not just on the ground plane. A side-by-side comparison of a pointcloud from a stationary LiDAR sensor in simulation against the estimated foreground is shown in Figure 2.5

2.3.2 Background Segmentation in Stationary Applications

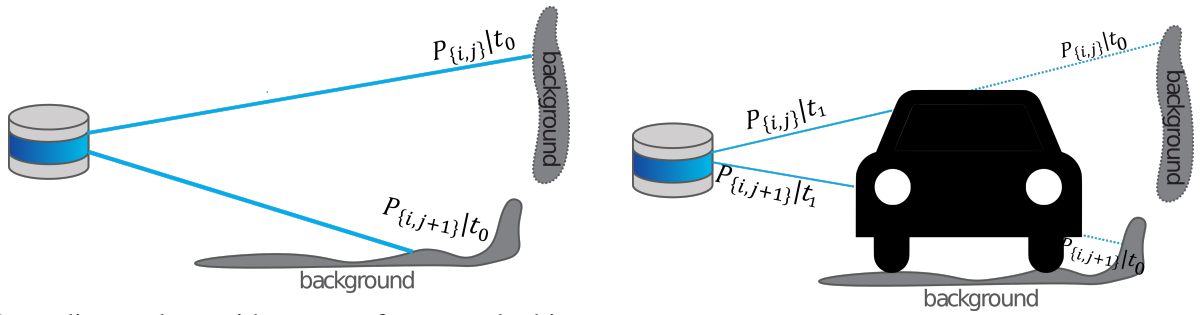
In a moving LiDAR application, background segmentation must be performed dynamically to distinguish moving objects from the background. However, with LiDAR measurements taken within a fixed location, removing both infrastructure and ground points need not change in process at each time step.



(a) Pointcloud with the background and foreground

(b) Pointcloud with the estimated foreground

Figure 2.5: Example of background removal - points in a red translucent bounding box are true foreground points



(a) readings taken without any foreground objects present

(b) readings taken with foreground object present

Figure 2.6: Two-frames of measurements taken during the max azimuth-channel distance method

2.3.3 Max Azimuth-Channel Distance Method

A common methodology in stationary LiDAR applications is the "Max-Distance" [8] or "Azimuth-Channel-Distance" [9] method. Here azimuth refers the horizontal scanning angle of the LiDAR, while channel refers to the beam number. In this procedure, the background is filtered based on maximally recorded distances for each azimuth angle and channel of the LiDAR. Because objects in the foreground will have less distance to the background, any points that are less than this maximal distance are kept. This process though assumes that the background is in-penetrable, and though this is generally a safe assumption, objects such as trees, guard rails, and fences may not meet this assumption due to small variations in the LiDAR's scanning pattern and or wind.

Figure 2.6 illustrates a scenario of readings of two beams at $t_{0,1}$. During the first frame t_0 distances $P_{i,j}, P_{i,j+1}$ are stored in a $I \times J$ matrix (or lookup-table) where I represents the total number of azimuth angles and J represents the total number of LiDAR channels. Subsequently at the next time-step t_1 beams $P_{i,j}, P_{i,j+1}$ are incident on the vehicle, and their distances will be lesser than the values stored in the look-up table. Correspondingly, these points from these two beams are kept as they are assumed to be foreground points.

Note that the following example only sampled for two frames; in an actual implementation, one would sample for multiple frames to converge on an initial estimate for the max-distance background matrix.

With the max azimuth-channel method, the following assumptions are held:

1. The background is impenetrable;
2. LiDAR pointclouds are measured in a fixed-pattern;
3. A background point exists for every beam and azimuth angle of the LiDAR. Alternatively, if no points are read for a beam and azimuth angle, then any returns from that beam-angle pair can be assumed to be from the foreground;
4. Dynamic objects move sufficiently during the sampling period.

If these assumptions are violated, then the background filtering method may not work as intended.

2.3.4 Azimuth-Channel Occupancy Map

The next two methods for background removal can be thought of time-based cell occupancy methods. In the Azimuth-channel occupancy map method, structures from fixed-pattern LiDARs are leveraged to reduce the total cell count that would be required normally in a 3D representation.

In occupancy map concepts, pointcloud readings are binned into small cells, where multiple readings in a cell will increase the occupancy count. Because background points are likely to be occupied for a greater period of time than foreground points, a simple filtering method can be devised to remove readings of points whose corresponding cells have higher occupancy rates against the total number frames sampled.

In the azimuth-channel occupancy map, cells are split by azimuth angle, radius, and channel of the LiDAR. An illustration of measurements taken during three frames is shown in Figure 2.7. The cells containing blue and red points in Figure 2.7 have been occupied for a greater time compared to cells with yellow points; in this example one could filter set an occupancy threshold to 66% to only keep the yellow cells.

2.3.5 3D Voxel Occupancy Map

The 3D Voxel occupancy map is similar to the prior method, except that no assumption is made for the LiDAR's scanning pattern. Instead, the world is divided into three-dimensional cells or

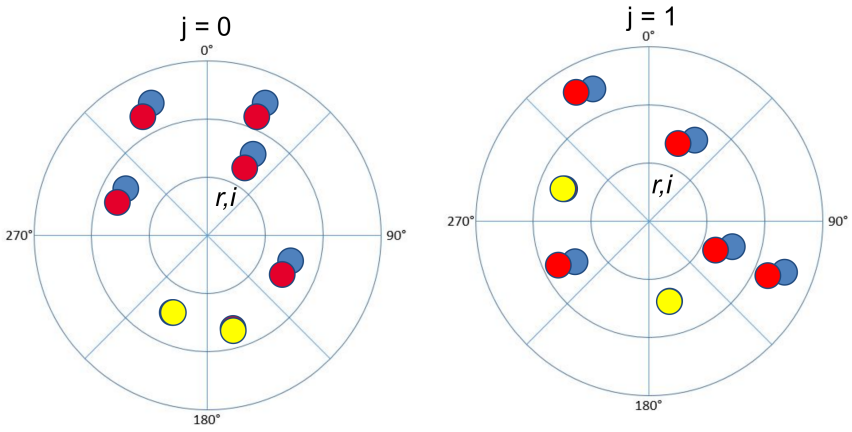


Figure 2.7: Azimuth Channel Occupancy map taken for three measurement frames on two channels of the LiDAR $j = 0, 1$

"Voxels". XYZ pointcloud readings are then binned into cells, where similar occupancy thresholds can be used to determine background points. For example, Figure 2.8 illustrates two-slices along the z axis of the voxel occupancy map taken during three measurement frames with a LiDAR. Cells that occupy a large amount of time (e.g. cells with blue and red points) can be filtered by again setting an occupancy threshold to 66%.

Due to small cell-sizes, occupancy-map methods require significantly more computation time and memory use. In practice, programmers optimize these methods by choosing hierarchical data structures such as KD-Trees [17] or OCTREEs [18] which can more efficiently represent sparse 3D Data.

The background method selected for areas without large amounts of vegetation is the Max Azimuth Channel Distance method due to its computationally low cost. If there are large amounts of vegetation in the scene, then either the Azimuth Channel 3D voxel occupancy map methods can be used due having more tolerances the noise in vegetation.

2.4 3D to 2D PointCloud Projections

With the estimated background removed, the next step would be to perform clustering over foreground points to identify objects. However, prior to clustering, the dimensionality of the point-cloud data can be reduced by creating 2D projection views which may be more intuitive to cluster

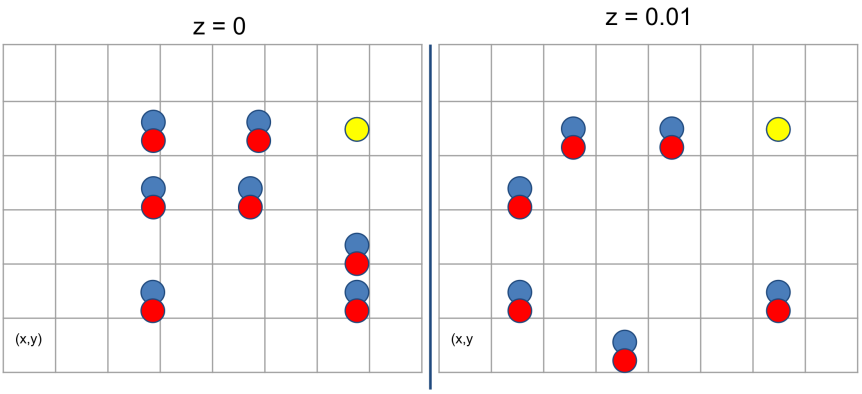


Figure 2.8: 3D Voxel Occupancy map taken for three measurement frames on two slices of the occupancy map $z = 0, 0.01$

on.

2.4.1 Side View Projections

Side-view projections are taken in the YZ plane, as show in Figure 2.9. In $360°$ LiDARs, a side-view projection can also be panoramic projections. Side projections can be problematic though as objects may overlap with one another – for example two vehicles could overlap if they are in adjacent lanes.

2.4.2 Bird's Eye View Projection

A top-down (XY), or Bird's Eye View (BeV) projection can be though of looking at the point-cloud from the perspective of a bird flying overhead, as illustrated by Figure 2.9. Unlike the side-view projection, it is generally safe to assume that moving objects on the ground should not overlap with one-another traveling in the (XY) plane.

2.4.3 Image Conversions

In subsequent sections, standard computer vision image operations will be utilized in the clustering and detection processes. To utilize standard image operations found in common programming libraries, the pointcloud projection can be converted into a bitmap image that is scaled where each pixel correlates to 2D XY position. Equation (2.7) details a combined projection and image

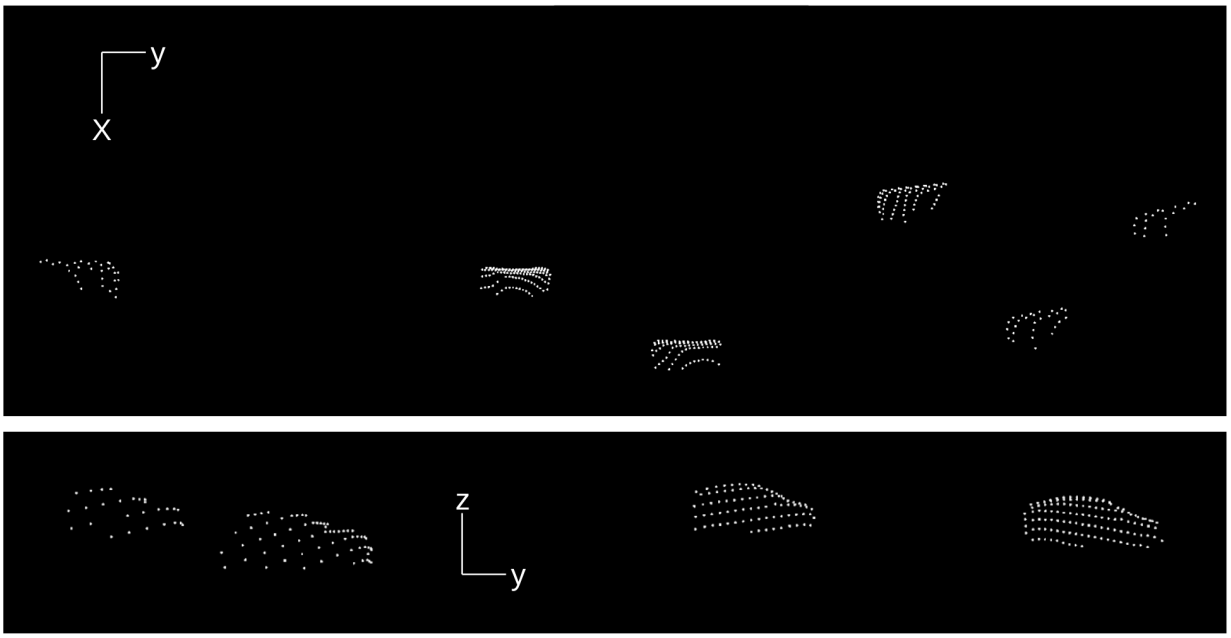


Figure 2.9: LiDAR Projection views in the XY (upper figure) and YZ (lower figure) planes

translation process from a 3D pointcloud:

$$I(px, py)^{m \times n} \in \{0, 1\}$$

where:

$$px = int_{16}((x + x_{min}) * scale + 0.5)$$

$$py = int_{16}((y + y_{min}) * scale + 0.5) \tag{2.7}$$

Hence, even with multiple point pairs (x, y) at different heights z, the bitmap will still remain 1. It is recommended that a limit $x_{min} \leq x \leq x_{max}$, $y_{min} \leq y \leq y_{max}$ be applied to bound and preallocate the image dimensions $m \times n$. The image scale can be empirically chosen; through trial and error, it was found that a scale of 8 to 10 worked best for a 15×60 m area [1]. The scaling factor affects the precision of the data - a scale factor of 10 pixels per meter will have a precision of $\pm 10cm$ [1].

2.5 Clustering

Once the background is removed, and a BeV projection is performed, the next step in the object detection process is to identify clusters of points such as ones in seen from LiDAR readings on vehicles in Figure 2.9. The section continues with a survey with clustering methods commonly used, and criteria developed to select a clustering algorithm.

2.5.1 Survey of Clustering Algorithms

A quick survey of clustering algorithms taken show many different methods, as seen from clustering algorithms available in the popular python scikit-learn package[2] shown in Figure 2.10. To aid in selection, a list of criteria is developed for selection:

- Number of clusters need not be known prior to clustering;
- Square / rectangular densities of points are classified as a single cluster, rather than multiple;
- Every point need not be part of a cluster, and instead could be noise in the data;
- Execution time is $<$ 100ms. This time is selected for a LiDAR update rate of 10 Hz.

Table 2.1 summarizes the criteria against the clustering methods shown in Figure 2.10. As can be seen, the only method (among the surveyed) that meets all of the listed criteria is the Density Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [19].

2.5.2 DBSCAN

DBSCAN has been a popular method for clustering LiDAR data [20] [9]. The clustering algorithm contains two tunable parameters ϵ and η_{min} which describes the maximum spacing between points in a cluster, and minimum points for a cluster to be formed, respectively. Figure 2.11 illustrates an example of data that is clustered and the different types of points. In the figure, points that are colored red are core points, which have more than one connected node by minimum distance ϵ. Points that are colored yellow are still cluster points, but are only connected by one node point up to ϵ distance away. Lastly, points that are not within ϵ distance away from any core points, and

Table 2.1: Comparison of clustering methods available from scikit-learn [2]

Clustering Method	**# of clusters not known**	**Square densities clustered as as one cluster**	**Execution time <100 ms**
KMeans	✗	✗	✓
Affinity Propagation	✓	✗	✗
MeanShift	✓	✓	✗
Spectral Clustering	✓	✗	✗
Ward	✓	✗	✗
Agglomerative Clustering	✓	✓	✗
DBSCAN	✓	✓	✓
OPTICS	✓	✓	✗
BIRCH	✓	✗	✓
Gaussian Mixture	✓	✗	✓

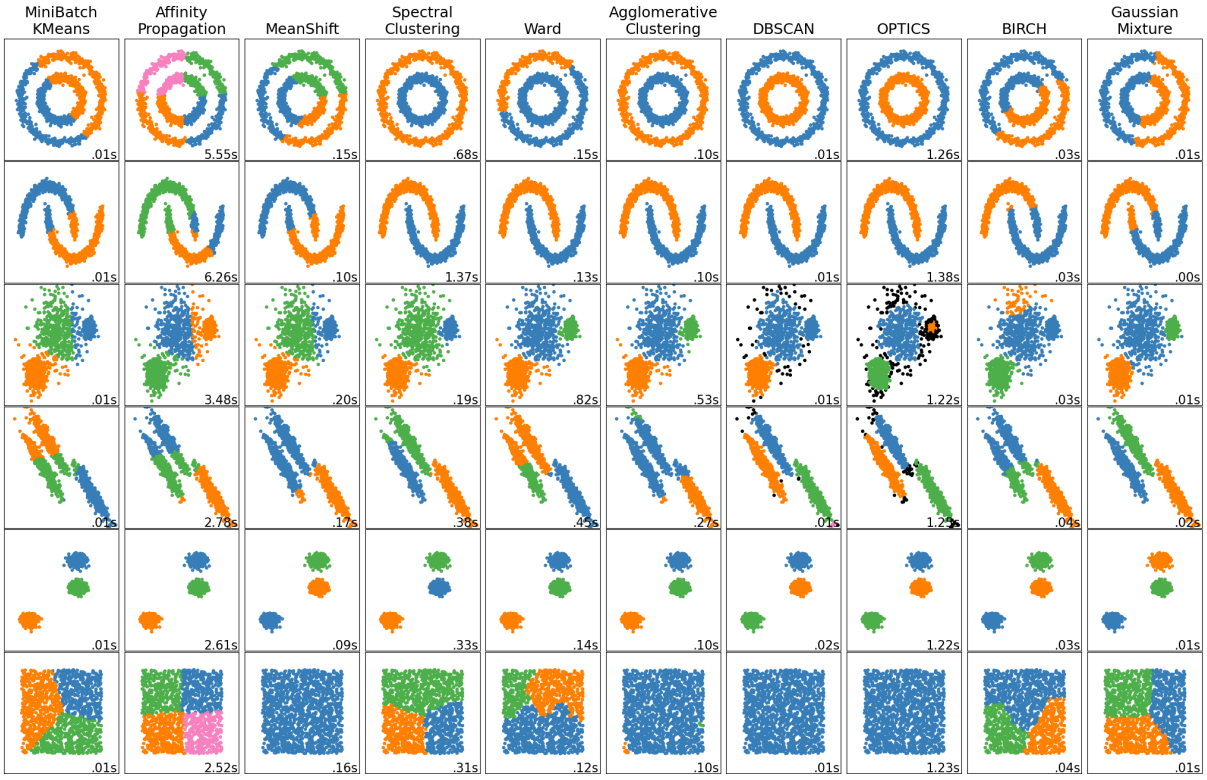


Figure 2.10: Survey of Clustering Methods Available from the Python library scikit-learn [2]

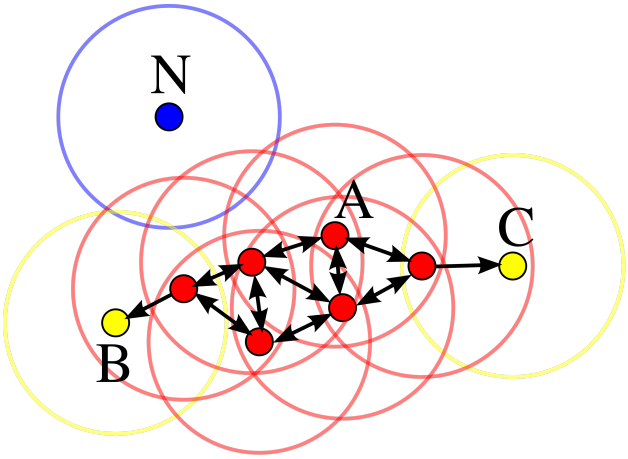


Figure 2.11: Three types of points in the DBSCAN algorithm.[3]

cannot be formed a new cluster with at least η_{min} points, are classified as noise points (colored blue in the figure). DBSCAN's main advantage is that points that can classified as noise points, which significantly helps prevent unremoved background points from being a part of a cluster. However, the LiDAR's characteristic of returning sparser pointclouds as the radial distances increases makes it difficult to choose a single ϵ value. Zhao et. al. proposed a variable ϵ based on the radial distance from the LiDAR [20]. For instance, pointcloud readings such as one shown in Figure 2.12 may have a portion of the object truncated during clustering due to the ϵ and η_{min} settings. Although the ϵ parameter can be tuned to achieve better clustering in Figure 2.12, through experimentation in previous work [1], it was found that a image dilation pre-operation on the image before clustering yielded better overall clustering results.

2.5.3 Dilation Convolution Operations

A dilation convolution is a standard image morphological transformation technique that expands the outline of pixels, and tends to fill gaps in spaces. Figure 2.13a-b illustrates the change in pixel density after a single 3×3 dilation kernel of ones. Although the dilation kernel size, value, and number of operations can vary, [1] found that a single operation of a 3×3 dilation kernel of ones was sufficient.

When using a dilated image for clustering, the bounding boxes for clusters are slightly larger

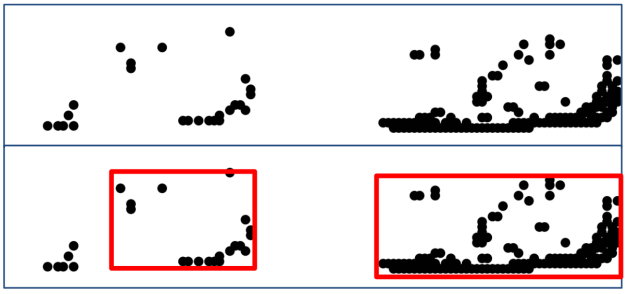


Figure 2.12: DSBSCAN Clustering on a undilated BeV LiDAR image with two vehicles. The left cluster is undesirable due to the sparsity of the data and under-clustering.

than the true bounding box due to the dilation operation expanding the outline. Hence, it is desirable to then to search on the original undilated image with the dilated bounding boxes to obtain the slightly smaller true bounding boxes. Visually, this process can be seen in Figure 2.13, or in pseudocode in Algorithm 1.

Algorithm 1: Vehicle Detection from LiDAR BeV bitmap [1]. ©2021 IEEE.

Input: Dilation Kernel w_k, convolution ops η_d
Data: BeV LIDAR bitmap $f(x,y)^{m\times n} \in \{0,1\}$
Output: Detections: [Pos. c_x, c_y, Dimensions w, h]
$P \in \mathbb{R}^{mn\times 2} \leftarrow$ `ImgToPts(`$f(x,y)$`)`
for $i \leftarrow 1$ ***to*** η_d **do**
  $f(x,y) \leftarrow w_k * f(x,y)$;
end
$P_g \in \mathbb{R}^{mn\times 2} \leftarrow$ `ImgToPts(`$f(x,y)$`)`
$b_{boxes}, clusters \leftarrow$ `DBSCAN(`P_g , ϵ , n_{min}`)`
for $i \leftarrow 1$ ***to*** *len(clusters)* **do**
  if `Classify(`*clusters[i]*`)` ***is*** *vehicle* **then**
    `// use undilated points`
    $b_{boxes}[i] \leftarrow$ `RefineBbox(`$b_{boxes}[i], P$`)`
  else
    $b_{boxes}[i]$`.del()`
  end
end

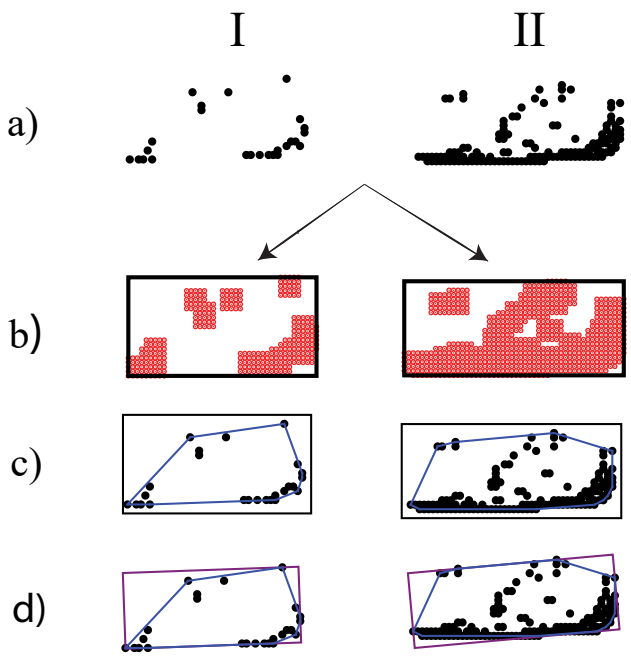


Figure 2.13: Vehicle Detection Process: (a) Bird's Eye View LiDAR bitmap, (b) Dilated image with DBSCAN generated bounding boxes, (c) bounding box search on original data, and (d) oriented bounding boxes as the final output [1]. ©2021 IEEE

2.6 Classification

The methodology for classification in this thesis is not extensively studied, in part due to the large amount of annotation efforts required. Instead, rule-based methods similar to [8] are employed. Detections are binary classified into vehicle and non-vehicle, based upon the area of the detection.

2.7 Orientation Detection

After classification occurs between vehicle and non-vehicle, an oriented bounding-box is more desirable for vehicles as they often change in orientation. To develop an estimation on the orientation of a vehicle, concepts from [14] for orientation detection are applied. In this work's approach, a weighted cost-function is developed based on candidate orientations for the bounding box with two criteria. Subsequently, the minimal cost orientation is then selected.

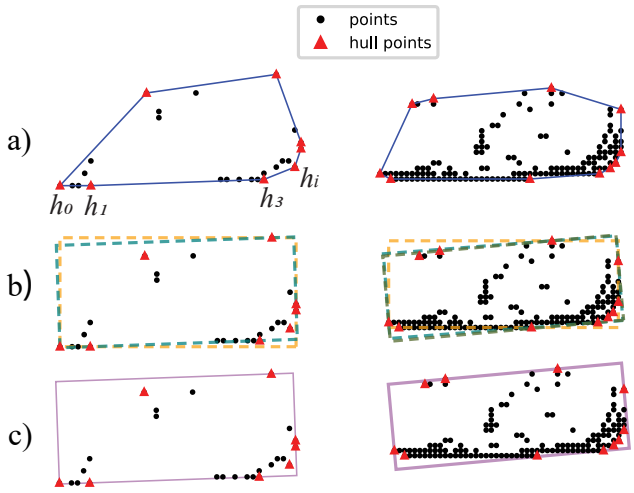


Figure 2.14: Orientation selection process based on candidate bounding boxes: (a) Convex hull creation with vertex points h_i, (b) candidate bounding boxes generated using angles between h_i, and (c) final selection based on cost-minimization of area and edge to vertex distances.

2.7.1 Orientation Cost Function Generation

The selected criteria for orientation cost fitting include:

1. The oriented bounding box area - the smallest area bounding box may be most in-line with the edges of the vehicle;

2. Distances from the vertices of the ConvexHull that forms the cluster to the edges of the oriented bounding box - vertices of the polygon may be incident on the oriented bounding box.

Equal weighting is used between the two criteria. The process for orientation can be seen visually through Figure 2.14 or algorithmically in Algorithm 2.

2.8 Object Tracking

Object tracking involves the process of spawning new tracks, associating detections with existing tracks, reconciling previous track history with new detections, and destroying old or lost tracks. The process flow diagram where detections are stored into tracks is illustrated by Figure 2.16.

Algorithm 2: Refine Bbox Algorithm

Input: Cluster of Points $P_c \in \mathbb{R}^{n\times 2}$, $b_{box} \in \mathbb{R}^{4\times 2}$
Output: Oriented bounding box: [c_x, c_y, w, h, θ]
$C_{vtcs} \in \mathbb{R}^{v\times 2} \leftarrow$ `ConvexHull(`P_c`)`
$\theta_s \leftarrow$ `Unique(HullAngles(`C`))`
for $s_i \leftarrow 1$ ***to*** *len(s)* **do**
  $RB_{box} \in \mathbb{R}^{4\times 2} \leftarrow$ `RotateBBox(`θ_{si}, b_{box}`)` $cost(s_i) \leftarrow w_1$ `Area(`RB_{box}`)`
  $cost(s_i)$ $+= w_2$`Sum(DistToEdges(`RB_{box}, C_{vtcs}`))`
end
$ret \leftarrow \theta_s($ `argmin(`$cost$`)` $)$

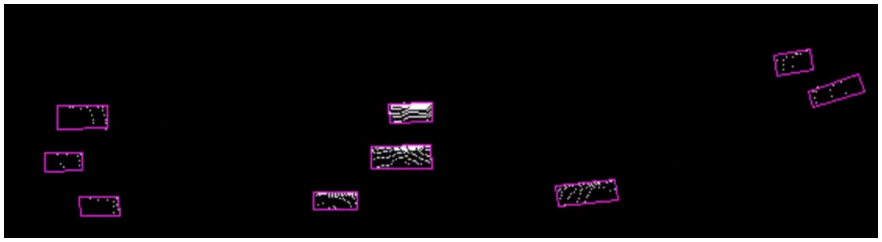


Figure 2.15: Example of detections in simulation with oriented bounding boxes

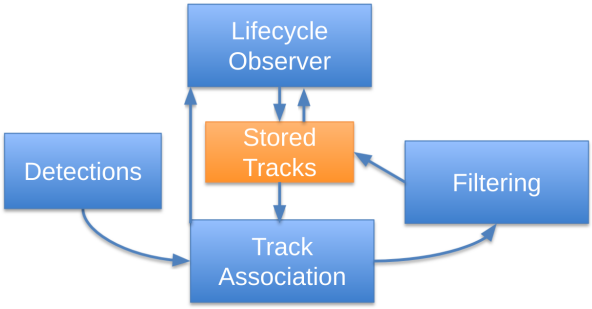


Figure 2.16: Object tracking flow diagram

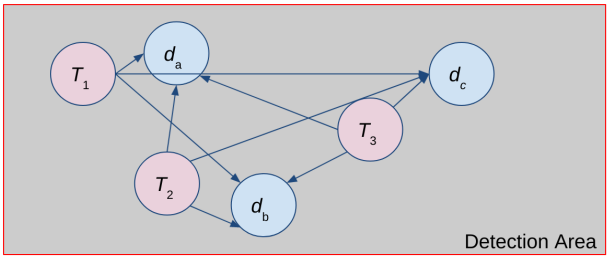


Figure 2.17: Pair-wise relations between new detections and existing tracks shown by arrows in a detection area

2.8.1 Track Association

Because objects may move in many different motions, new object detections need to be associated with current tracks in the system to maintain a state-history for an object. Two concepts for data association between detections and tracks are presented, with the first method implemented for this work.

2.8.1.1 Single Assignment Methods

In single assignment methods, each detection is hypothesized to correlate up to one track at a maximum. Note that this does not mean that every detection must correlate to a track, as detections may also form new tracks instead. Under this hypothesis, it is then desirable to: (1) determine which detections are to be correlated with existing tracks, (2) which detections are of objects not yet tracked in the tracking module, and (3) which tracks that are missing a detection.

To accomplish this, a pair-wise cost function is usually formed between all detections and tracks, as illustrated in Figure 2.17. The pair-wise cost can be based on multiple metrics, including:

- Euclidean distance between the track and detection;
- Feature similarity (e.g. image similarity);
- Velocity similarity;
- Probability based on location (e.g. entrances and exits from the detection area).

In this work, Euclidean distance is selected for generating the pair-wise cost function. Observing the cost-function in the case where there are equivalent detections and tracks, a matrix relating each cost for assigning a detection to a track can be formed – such as one seen in Figure 2.18a. As previously noted, up to one detection can correspond to a track; therefore, the desire is to minimize the cost matrix in Figure 2.18a such that the total cost of assigning each detection to a track is minimized. To achieve this, the improved Hungarian algorithm [21] is utilized to optimally pair tracks with new detections. Note that in Figure 2.18a, choosing to pair D_b with T_1 has a lower cost than with pairing of D_a; however, the total cost of the system is still minimized with the selection.

In cases where there are less detections than tracks, such as Figure 2.18b, at least one track will be missing a detection. In this case, a tracking filter (discussed in the subsequent sections) is utilized to predict motion of the object at the for the next timestep. Inversely, there may be more detections than tracks such as the case in Figure 2.18c. Under these conditions, at least one new track must be formed and is sent to the lifecycle observer (also discussed in subsequent sections).

Lastly, a special case arises when there are equivalent detections and tracks, but the cost for associating some detection(s) to track(s) may be above a threshold limit. The scenario in which this may happen is when a object detection from a yet-to-be tracked object is received simultaneously while a detection for a currently tracked object is missing. It is then not desirable to associate the new object with an old object, and thus this track and detection must be sent to prediction, and the lifycycle observer, respectively. This special case is illustrated by Figure 2.18d, while an algorithmic description for entire the detection to track single-assignment association process is shown in Algorithm 3.

Algorithm 3: Tracking and Filtering Algorithm

Input: Currently Tracked Objects T_{objs}, new Detections D_{objs}
Output: Updated Tracks $\hat{T}_{\text{objs}}$, Unssigned Tracks UT_{objs}, Unassigned Detections UD_{objs}
$UT_{\text{objs}} \leftarrow T_{\text{objs}}$
$UD_{\text{objs}} \leftarrow D_{\text{objs}}$
$C_{\text{cost}} \leftarrow$ `EuclideanCostCalculator(`T_{obj}, D_{objs}`)`
$C_{\text{cost}} \leftarrow$ `DropRowColumnsMaxCost(`$C_{cost}, MaxCost$`) /* removes rows and columns where every element is > max-cost */`
$D_{ids}, T_{ids} \leftarrow$ `LinearSumAssignment(`C_{cost}`) /* indices of Linear Sum Assignment Optimization */`
for $D_{id}, T_{id} \in D_{ids}, T_{ids}$ `// every detection corresponds to a track` **do**
  $\hat{T}(T_{id}) \leftarrow$ `FilterUpdate(`$D(D_{id})$`)`
  `del(`$UT_{objs}(T_{id})$`) // Delete updated track from unassigned list`
  `del(`$D_{objs}(D_{id})$`) // Delete used detection from unassigned list`
end
$ret \leftarrow \hat{T}_{\text{objs}}, UT_{\text{objs}}, UD_{\text{objs}}$

	D1	D2	D3
T1	0.6	0.5	8
T2	3	0.5	2
T3	7	4.0	0.7

(a) Equivalent number of detections and tracks.

	D1	D2
T1	1.0	1.1
T2	13.0	0.9
T3	0.5	4.0

(b) Number of detections is $<$ than the number of tracks

	D1	D2	D3
T1	1.0	1.1	1.2
T2	13.0	1.0	1.5

(c) Number of tracks are is $<$ than the number of detections

	D1	D2	D3
T1	1.0	8.0	5.0
T2	13.0	3.0	0.8
T3	3	4.0	20.0

(d) Equivalent number of detections and tracks, but the cost between $T3 - D3$ is too high

Figure 2.18: Four cases of detection to track assignments. Values in the table represent the cost to association each detection to a track. Costs that are highlighted yellow show are for detection-track pairs that minimize the total assignment cost.

2.8.1.2 Multiple Assignment Methods

Multiple assignment methods, such as the PDAF or JPDAF algorithms [22], do not constrain only matching up to one detection to a track. Instead, it may be hypothesized that multiple detections could correspond to the same track, and subsequently may use multiple detections to update the history of the track. Although initially non-intuitive, multiple assignment methods are often more robust in handling cases where multiple partial detections are received for the same object. Traditionally, these methods improved tracking methods where the sensing technology contained multiple detections for an object (e.g. RADAR or SONAR). However, the methodology in this work does not investigate multiple assignment methods, as the clustering and detection process for LiDAR readings are less at risk for split readings.

2.8.2 Filtering and Motion Models

Detection Filtering in this section refers to the process in which associated measurements are reconciled with historic track data. Because measurements are often noisy, it would be undesirable to fully accept new sensor readings, especially if there are priors. Instead, a model of the predicted motion behaviour can be used to reduce measurement noise. Further, in timesteps where there are no object measurements (e.g. due to occlusion), the object motion is still able to be predicted.

2.8.2.1 State Space Representations

State space equations of systems describe the evolution of certain properties in a model either from some input and or time. For instance, and object in a 2D world may have the following state-space:

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \tag{2.8}$$

where x and y are the 2D coordinate position of the object, and v_x / v_y are the speed of that object in an inertial frame. Modeling how dynamic systems evolve is an historic area of research study. More complex system models require advanced computation, and accurate information or parameters of the system. In general, a linear dynamic system's continuous time evolution with an input $u(t)$ is governed by Equation (2.9):

$$X(t+1) = AX(t) + BU(t) \tag{2.9}$$

Where A describes the state transition matrix from the previous state to the next state, and B relates the control-input's state (U) to the system state (X). Continuous linear time systems require a infinitesimally sampling time; however, often measurements are taken in much larger intervals. Instead, the continuous linear system is discretized in timesteps based on the sensor update frequency, as show in Equation (2.11):

$$X(k+1) = AX(k) + BU(k) \tag{2.10}$$

If the inputs to a system are not known, models can still be developed; however, the control input $U(t)$ is instead set to zero:

$$X(k+1) = AX(k) \tag{2.11}$$

Without the input known, the only missing information for developing a model is the A matrix. The next few sections will describe system models that can be used for a moving object when the control input is not known.

2.8.2.2 Constant Velocity Models

The constant velocity model for an object in an inertial frame can be expressed as a set of linear equations with a sampling time dt:

$$x_{k+1} = x_k + v_x dt \tag{2.12}$$

$$y_{k+1} = y_k + v_y dt \tag{2.13}$$

$$v_{x,k+1} = v_{x,k} \tag{2.14}$$

$$v_{y,k+1} = v_{y,k} \tag{2.15}$$

Or in a state-space matrix form:

$$X_{k+1} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_k \tag{2.16}$$

As the name implies, constant velocity models assume that an object is traveling at a constant velocity. If objects rapidly change in velocity, this modeling decision may no longer hold.

2.8.2.3 Constant Acceleration Models

The constant acceleration model instead assumes that objects are traveling at a fixed acceleration. Velocity can change through this model - though only at fixed rates. Similarly, the constant acceleration model for an object in an inertial frame can be expressed as a set of linear equations:

$$x_{k+1} = x_k + v_x dt + \frac{1}{2} a_{x,k} dt^2 \tag{2.17}$$

$$y_{k+1} = y_k + v_y dt + \frac{1}{2} a_{y,k} dt^2 \tag{2.18}$$

$$v_{x,k+1} = v_{x,k} + a_{x,k} dt \tag{2.19}$$

$$v_{y,k+1} = v_{y,k} + a_{y,k} dt \tag{2.20}$$

$$a_{x,k+1} = a_{x,k} \tag{2.21}$$

$$a_{y,k+1} = a_{y,k} \tag{2.22}$$

Or in a state-space matrix form:

$$X_{k+1} = \begin{bmatrix} 1 & 0 & dt & 0 & 0.5dt^2 & 0 \\ 0 & 1 & 0 & dt & 0 & 0.5dt^2 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix}_k \tag{2.23}$$

Both constant velocity and constant acceleration models were tested in simulation; the constant acceleration model improve simulations where vehicles had more changes in velocity.

2.8.2.4 Kalman Filters

Kalman filters [23] are useful filtering algorithms that can combine a linear state-space model (which may have uncertainty) of a system with measurements taken (which may be noisy) to form a filtered output of an objects state. A large amount of existing literature can be found on theory for Kalman filters, and instead this section will describe only the measurement models and estimation methods used for process noise and measurement uncertainty.

The measurement matrix H relates a measurement to the corresponding state-space of the

system. E.g., $\hat{X} = Hz$ where z are the raw measurements, and $\hat{x}$ is the measured state. Often measurements correspond non-linearly with the system's state, however, detections can be represented by 2D centroid positions (x, y) for which the measurement corresponds exactly to the state-space models developed.

For the 4-state constant velocity the model, the measurement matrix H is given by:

$$H_{\text{cv}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.24}$$

While in the 6-state constant acceleration model, the measurement matrix H is given by:

$$H_{\text{ca}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.25}$$

The Kalman filter also utilizes an estimated noise matrix for sensor readings. Values were selected based on position errors collected in simulation for raw detection values:

$$R = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix} \tag{2.26}$$

Also exists is the process noise covariance matrix Q, which describes the overall noise and covariances in the system model. A common method for selecting the process noise is through trial-and-error. Another method for choosing a process noise matrix can be developed based on the sensor update rate [24]:

$$Q = \sigma \begin{bmatrix} \frac{dt^6}{36} & \frac{dt^5}{12} & \frac{dt^4}{6} & \frac{dt^3}{6} \\ \frac{dt^5}{12} & \frac{dt^4}{4} & \frac{dt^3}{2} & \frac{dt^2}{2} \\ \frac{dt^4}{6} & \frac{dt^3}{2} & dt^2 & dt \\ \frac{dt^3}{6} & \frac{dt^2}{2} & dt & 1 \end{bmatrix} \tag{2.27}$$

where σ is a tunable variance parameter, currently set to 0.1.

2.9 Track Lifecycle Management

Track lifeycle management refers to the process of creating new tracks from detections, and deleting old tracks. A track usually contains additional information besides the state-space model. For instance, a track object could contain the following:

- State of the object's model;
- Prior states of the object's model;
- Bounding-box history of the object's model;
- Total count of times the object received a detection;
- Total count of times the object did not receive a detection;
- Consecutive count of times the object did not receive a detection.

With additional information for tracks in the tracking module, creation and deletion criteria can be developed. For instance, typically a track is deleted if the total number of times it has be consecutively invisible is above a threshold. Likewise, a track could publish a confidence value based on the number of times the object has received a detection.

In the implementation of the life-cycle module, tracks are deleted if they have been consecutively invisible for more than 5 frames, or if they are approaching the edge of the detection area.

2.10 Algorithm Implementation

All algorithms are implemented in the Python programming language for use in simulation [1]. For experimentation, a separate code base was developed in prior work [1] that used a mixture of C++ and Python.

[1] Available through GitHub https://github.com/amirx96/stationary-lidar-object-tracking

3. EXPERIMENT METHODOLOGY AND METRICS EVALUATED

This chapter covers experimental designs, both in simulation and experimentation, as well as the metrics evaluated for object detection and tracking.

3.1 Experimentation

For experimentation, two previous datasets from [1] are utilized for primarily for vehicle detection. These datasets, which are also publicly available for other researchers[1], were collected with a VLP-16 sensor, mounted at 2.2 meters high on a portable tripod in two different roads. The first dataset, named TAMU_01, was collected on a 6 lane urban road segment, though the primary region of interests covers one travel direction (three lanes). The second dataset, named TAMU_02, was collected on a 4-lane highway speed segment, with again the primary region of interest covering one travel direction (two lanes). Quantitative analysis of TAMU_01 and TAMU_02 is made available through hand annotated LiDAR data of several hundred frames with vehicles in-scene.

An additional and new dataset is also collected in this thesis, named TAMU_03, with an emphasis for pedestrian detection, though only qualitative analysis is performed due to the annotation efforts required. In this dataset, a VLP-32c sensor is instead utilized for data-collection (shown in Figure 3.1), as pedestrian targets required a higher resolution LiDAR. LiDAR data from pedestrians was collected on Ireland street at Texas A&M University in College Station Texas during a class change when a large volume of foot-traffic was present.

3.2 Simulation

CARLA [25] is a freely available open-source simulator targeted towards researchers interested in a simulation tool for autonomous driving vehicle applications. Many of the traffic generation functionalities that is used for autonomous vehicle development can also be used for simulating traffic vehicles with a fixed LiDAR sensor. With having the ability to query each traffic vehicle precise location, velocity, and dimensions, the simulator provides for an easier method of ground-

[1] Available online through https://unmannedlab.github.io/research/Roadside-LiDAR

Figure 3.1: Experimental Setup for TAMU_03. A VLP-32c is mounted on a heavy-duty tripod, and connected to a 12v battery source. Ethernet is connected from the LiDAR to a laptop for data collection

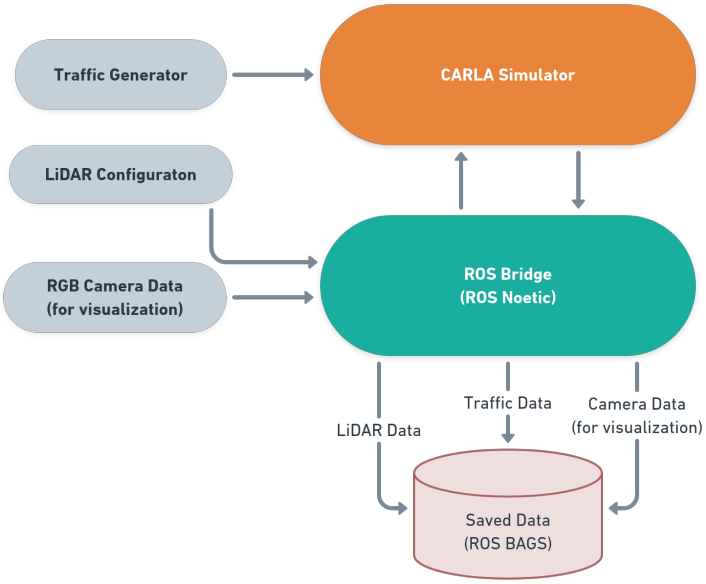


Figure 3.2: CARLA Simulator Framework Design

truthing and analysis on tracking performance compared to experimentation. The process flow for generating simulation data can be seen in Figure 3.2. The popular Robotic Operating System (ROS) is utilized for saving "bagfiles" of datasets, which can later be used for offline analysis.

3.2.1 Experimental Design

At the time of writing, the latest version (0.9.12) of CARLA is selected for use on Ubuntu 20.04 with a Nvidia GTX 1080 Ti. Two simulation datasets are developed with the CARLA simulator. The first simulation scenario (light traffic) covers a 3-lane highway segment under light traffic, with a curved road portion in the beginning. The second simulation dataset (stop-and-go) covers the same highway segment except with stop-and-go traffic instead.

For the simulation procedure, a traffic generation script procedurally spawns CARLA vehicles in three lanes just outside the LiDAR's FoV. Traffic then moves towards the LiDAR, as illustrated in Figure 3.3. Once a vehicle has moved well past the end of the LiDAR's detection range, the vehicle is deleted and subsequently a new vehicle is spawned at the original spawn point to maintain a constant volume of traffic. The amount of traffic can be controlled by changing the number of

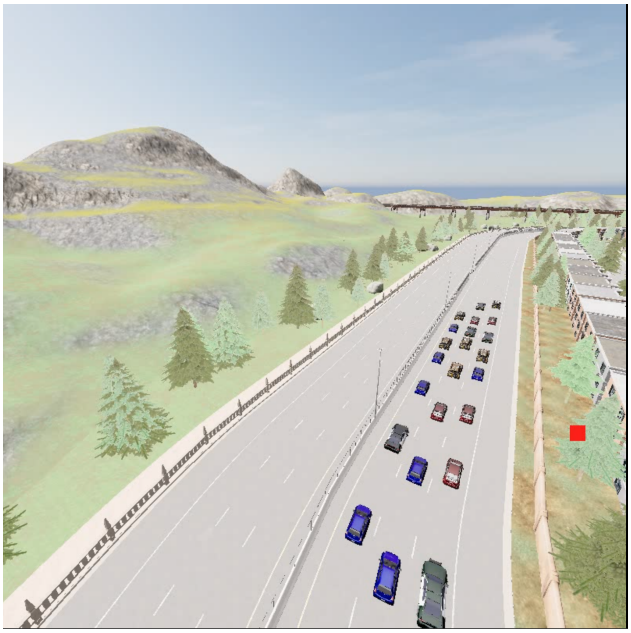


Figure 3.3: Example of traffic generated for a stop-and-got dataset. The LiDAR's location in this simulation is shown by a red square.

vehicles that spawn initially. For the stop-and-go traffic scenario, vehicles momentarily stop at a particular location causing traffic to back up (illustrated later in Figure 4.2).

3.3 Evaluation Methods

In order to quantitatively evaluate object detection and tracking methods, the analysis criteria must be defined. This analysis considers detection performance (false-positives, false-negatives), and tracking performance (using position, velocity, and IOU error). In this analysis, some metrics depend on the experimentation type, and ground-truth source. In simulation, we can easily extract ground truth data which is the number of moving objects in a scene, and each object's respective position, velocity, and heading. Compared to simulation, capturing ground-truth sources for experimentation have added difficulty. Most quantitative detection analysis resolve precision, recall, and mean intersection over union (mIOU) based on labeled sensor data. However, by utilizing the sensor data for labels, this may produce confounding results as the sensor data may not reflect the actual ground truth. For instance, the sensor Field of View (FoV) and occlusions from objects may not allow for any sensor readings on objects, and thus the evaluation based on labeled data cannot

capture this.

Instead, more robust quantitative evaluations utilize labeled data from other sensor sources with a reliable FoV, such as one presented in [26] where a helicopter captured aerial photography as ground-truth for detection, position, and velocity performance. Another robust method is one where a test vehicle equipped Real Time Kinematic (RTK) GPS, which allows for centimeter level accuracy, is used to evaluate position and velocity estimate from the LiDAR detection system.

3.3.1 Precision

Precision represents the false-positive rate and is the fraction of true-positives over the total number of detections. The calculation for vehicle precision in an individual single frame can be given by equation below:

$$P_{\text{vehicles}} = \frac{TP_{\text{vehicles}}}{TP_{\text{vehicles}} + FP_{\text{vehicles}}} \tag{3.1}$$

3.3.2 Recall

Recall represents the false-negative rate and is the fraction of correct vehicle detections over the true number of vehicles and any false negatives in a frame. The calculation for vehicle recall in an individual single frame can be given by equation below.

$$R_{\text{vehicles}} = \frac{TP_{\text{vehicles}}}{TP_{\text{vehicles}} + FN_{\text{vehicles}}} \tag{3.2}$$

3.3.3 Velocity Error

Velocity error is the hypothesized vehicle velocity over the ground-truth velocity. This metric is directly available for comparison in simulation, and is not available in experimentation. Velocity error can by analyzed component-wise 2-dimensionally, or as a magnitude:

$$v_{e|x}(t) = \bar{v}_x(t) - v_x(t) \tag{3.3}$$

$$v_{e|y}(t) = \bar{v}_y(t) - v_y(t) \tag{3.4}$$

$$V_e(t) = \sqrt{v_{e|x}(t)^2 + v_{e|y}(t)^2} \tag{3.5}$$

3.3.4 Position Error

Position error represents the distance error between the hypothesized area centroid and the true labeled area centroid of the object. In the case of simulation experimentation, position error is between the hypothesized centroid position and true centroid position of the object. Likewise, position error can be defined both in terms of components and magnitude, as shown in Equations (3.6) to (3.8).

$$\delta_x(t) = \bar{x}(t) - x(t) \tag{3.6}$$

$$\delta_y(t) = \bar{y}(t) - y(t) \tag{3.7}$$

$$\Delta(t) = \sqrt{\delta_x(t)^2 + \delta_x(t)^2} \tag{3.8}$$

3.3.5 Intersection Over Union

Intersection over union represents the area intersection between a hypothesized bounding box, and the ground-truth bounding box for a detection, divided by the area union of the two bounding boxes. Mathematically, it is represented by Equation (3.9) below. A visual example is also provided in Figure 3.4.

$$\text{IOU}(H, G) = \frac{H \bigcap G}{H \bigcup G} \tag{3.9}$$

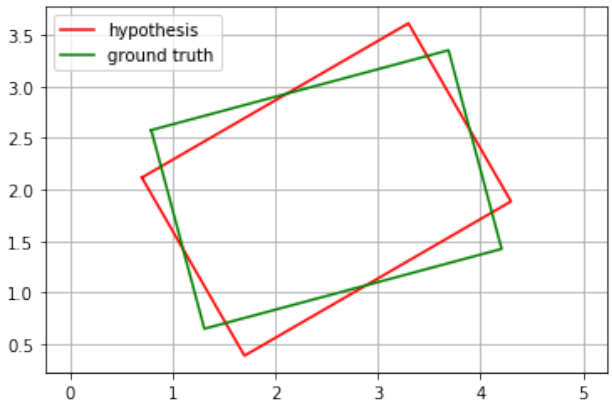


Figure 3.4: Example calculation of Area Intersection over Union: The IOU between the hypothesis and ground-truth in this example is 79.8%

4. RESULTS

The following chapter will cover results obtained in simulation and experimentation, according to the methodology developed in Section 3.3.

4.1 Experimentation

In experimentation, tracking results are derived from datasets previously published [1]. Precision, recall, and mean Intersection over Union are calculated against annotated LiDAR data. Table 4.1 details the experimental results for TAMU_01, and TAMU_02. For the pedestrian detection dataset (TAMU_03), qualitative analysis is performed in section 4.3.6.

4.2 Simulation

In simulation, precision, recall, velocity error, and position error are calculated against ground-truth data from the simulator. Results for two scenarios, "light traffic", and "stop-and-go traffic" are presented in Tables 4.2 and 4.3.

4.3 Discussion of Results

The next few subsections discuss performance of the multi-object detection and tracking system developed, and include analysis on impacts to performance. Similar precision and recall detection performances were obtained in both simulation and experimentation, suggesting that the simulations developed can compare to experimentation.

Table 4.1: Experimentation results for vehicle tracking derived from datasets in [1] ©2021 IEEE

Dataset (exp)	**Precision**	**Recall**	**mIOU**	**# of Vehicles**	**# of frames**
TAMU_01	99.5 %	93.9 %	0.59	**41**	546
TAMU_02	100 %	96.3	0.67	**13**	549

Table 4.2: Simulation results for vehicle tracking derived from the CARLA simulation tool

Dataset (simulation)	Total Expected Measurements	Total Taken Measurements	False Positives	False Negatives
light_traffic	5090	4,856	109	161
stop_and_go	30085	28,850	383	1618

Table 4.3: Summarized Simulation Results

Dataset (simulation)	Precision	Recall	mIOU	# of Vehicles Detected vs. Actual	Pos Error (avg)[m]	Vel Error (avg)[m/s]
light_traffic	97.6 %	94.6 %	0.69	**37 / 37**	0.39	6.9
stop_and_go	98.7%	94.6%	0.70	**69 / 66**	0.32	1.4

4.3.1 Missed Vehicle Detections

In both experimentation and simulation, decent recall rates ($> 94\%$) were achieved, though not as high as precision. This follows intuition, as the system was more likely to miss detections rather than falsely identify detections that were not actually present. Analysis on generated video outputs showed a variety of reasons for missed detections. In simulation, detections were more likely to be missed due to more accurate ground-truthing methods compared to the annotation method used in experimentation. Missed detections in simulation could occur from sensor occlusion (e.g. the LiDAR's view was obstructed by a closer vehicle). In both simulation and experimentation, detections were also missed due to sparser readings for some vehicles causing the classification algorithm to not meet the threshold for a detection. In experimentation, it was also observed that darker colored vehicles often produced sparser readings due to low reflectiveness of the target.

4.3.2 Extraneous Vehicle Detections

In both experimentation and simulation high precision rates ($> 97\%$) were obtained, supporting that the developed system overall had a very low rate of false-positive detections. In the case of a false-positive, this did not always mean that a new and extraneous tracking objects were created.

Rather, individual frames may have contained an existing track that was not actually in-scene. This phenomenon primarily occurred during a track's exit of the detection area, where the Kalman Filter held the tracked object still in-scene occasionally until the max consecutive invisible count was reached.

In the stop-and-go scenario in simulation, an inverse effect was also observed. Vehicles exiting the detection area had low velocities, and in some instances tracks were too preemptively deleted due to their predicted centroid precision exiting the detection bounds. Because of their preemptive deletion, this led to new tracks being created for the same previously deleted track. In the stop-and-go scenario, this happened 3 times where 69 vehicles were estimated to have traveled through the highway, when only 66 had actually passed through.

Occasionally in experimentation, it was initially observed that noise from the unfiltered background could actually be large enough to occasionally form detections. These detections were later filtered out by implementing a rule-base classification system that removed smaller area detections.

4.3.3 Errors in Orientation

Orientation errors in the predicted bounding boxes were quite common in both simulation and experimentation, and is revealed in part by the mIOU metric (e.g. see Figure 3.4 for the IOU score with an orientation error). Figure 4.1 below illustrates an example of an erroneous orientation detection. Because of the cost minimization function, some orientations may have a minimal cost but still may not represent the desired bounding box. Vehicle 21 in Figure 4.1 had an incorrect orientation due to the vertices of the convex hull having closer distances to the edges of the misoriented bounding box.

4.3.4 Errors in centroid position

Position errors of up to $0.4m$ were observed in simulation. Often, centroid position errors could be linked to receiving partial detections on the object, causing the detected bounding box to be smaller and thus changing the calculated centroid. An example of this error can be seen in Figure 4.2 where Vehicle 6 changes in dimension due to having received a partial detection in

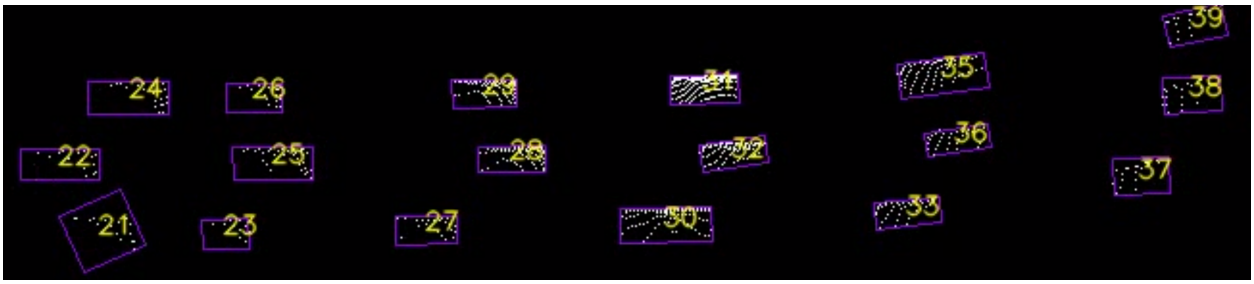


Figure 4.1: Example of a erroneous orientation for vehicle 21.

subsequent frames.

Comparing the light traffic and stop-and-go scenarios, the velocity of vehicles also appeared to had some effect on position errors. It can be hypothesized that faster moving objects may have greater margins for position error due to a fixed sampling and filtering time.

4.3.5 Errors in velocity

Errors in velocity were the largest relative to the other metrics observed. The cause for this can be associated to high position errors relative to the sampling time. Because the Kalman filter is incrementally updating estimates for the velocity, undesirable changes in centroid position are divided by a small sampling time leading to larger velocity errors. Interestingly, the stop-and-go scenario had a significantly lower velocity error. This may be attributed to instead the smaller average velocities for vehicles in this scenario.

4.3.6 Discussion on TAMU_03

Although no quantitative metrics were observed for TAMU_03, the detection performance can be qualitatively observed through Figure 4.3. It was observed that pedestrians were often over-clustered as one when the separation distances between them were small. This issue can be caused by: (1) improper tuning of the two parameters in DBSCAN, and or (2) the dilation operation conglomerating pedestrians from a top-down view into single objects.

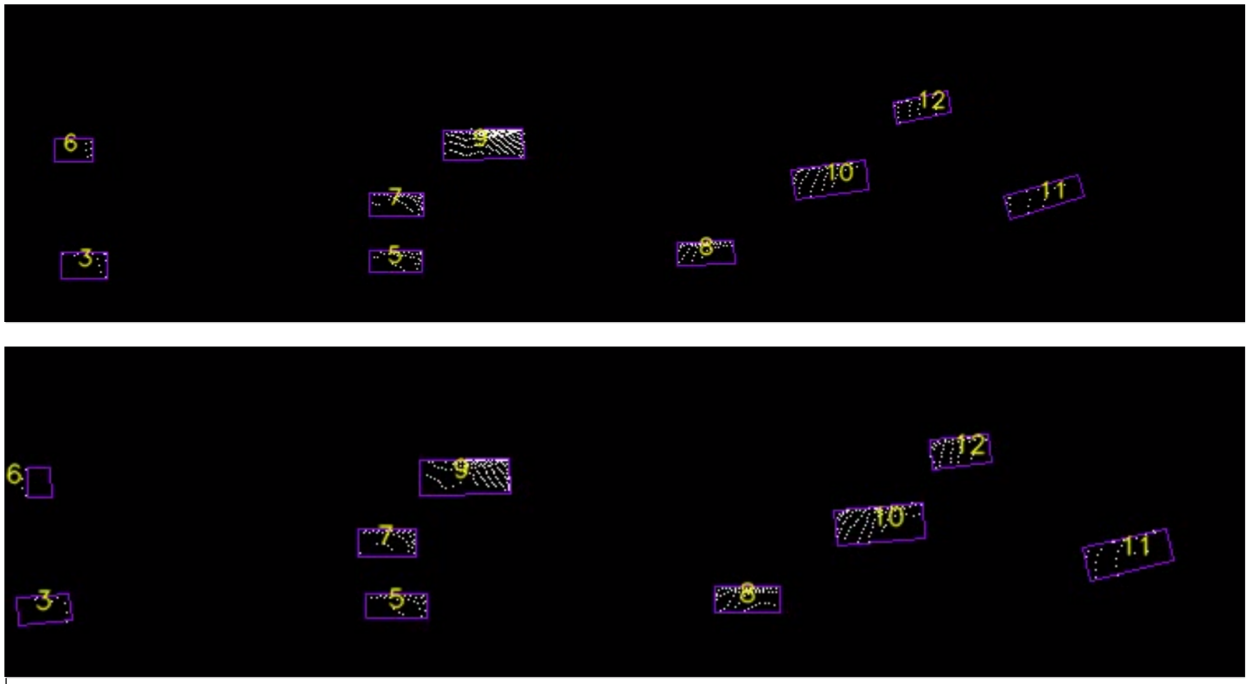


Figure 4.2: Visualization of the tracking algorithm at two different instances in time. Note the change vehicle 6's bounding box between timesteps.

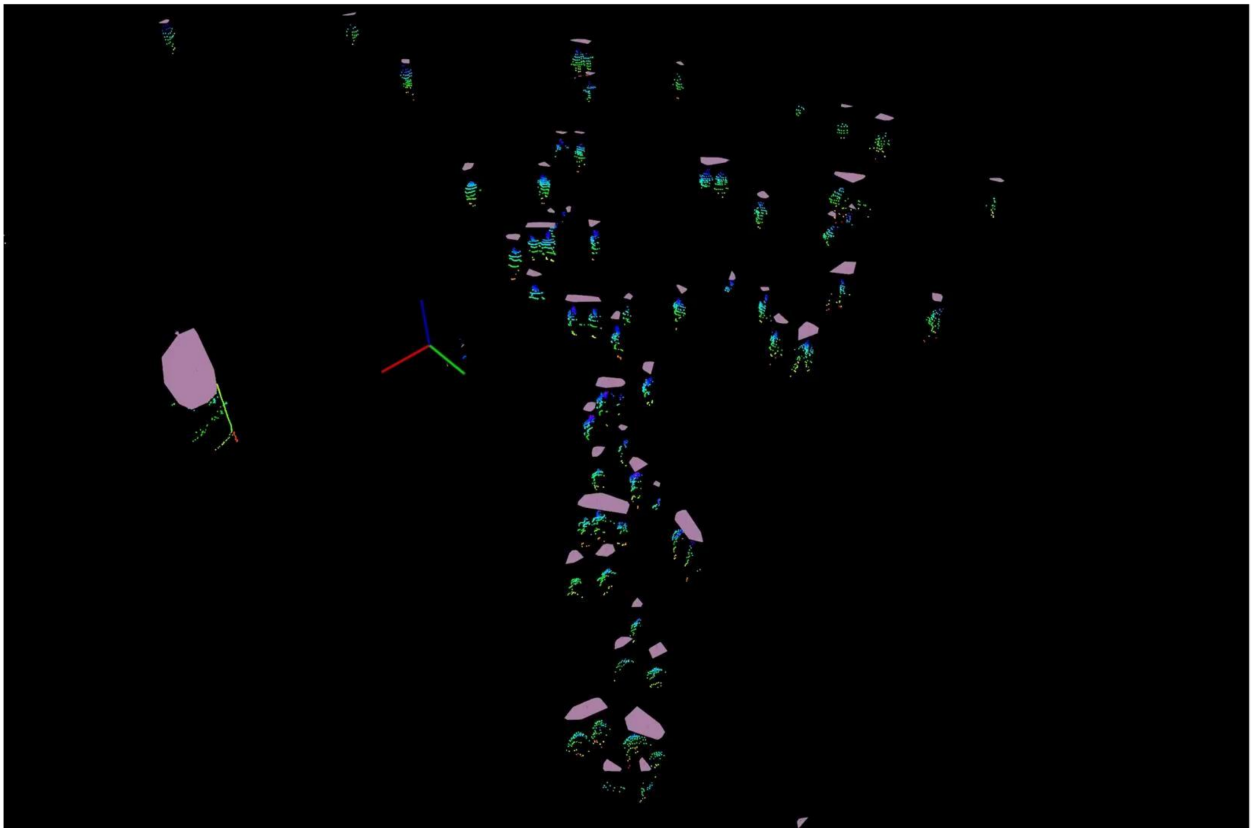


Figure 4.3: Pedestrian detection in TAMU_03. The foreground pointcloud is shown in an isometric view with overlayed 2D Polygons of detected pedestrians

5. SUMMARY AND CONCLUSIONS

The following chapter summarizes the methods used for a LiDAR based object detection and tracking system, results obtained from simulation and experimentation, and presents conclusions from this thesis. Lastly, future work is presented to aid any future researchers interested on the limitations and future areas of study for this work.

5.1 Methods Summary

For object detection from LiDAR data, a geometric feature-based approach was taken. This approach encompassed several stages, including (1) background estimation and removal, (2) point-cloud projection, (3) clustering, classification, and detection, and (4) tracking and filtering.

For background estimation and removal, three methods were detailed and comparisons were provided. Subsequently after removing the background, point-cloud projections in a Bird's Eye View were used to efficiently cluster the LiDAR data. As a pre-processing step to clustering, the projected LiDAR data was converted into a bitmap image, and a dilation kernel convolution was used to fill in the sparsity of the LiDAR returns. Next, the DBSCAN method was selected among the surveyed clustering methods due to its favorable execution time, and ability to handle noise points. For classification, a rule-based method for distinguishing between vehicle and non-vehicle was developed based on the area of the detection's bounding box. Detection included orientation refinement processes with a cost minimization function on proposed orientations for vehicles. Lastly, a single-assignment distance cost matrix was utilized for data association during tracking with a Kalman Filter used for filtering centroid measurements taken from the detected bounding-box.

5.2 Results Summary

Results are obtained both in experimentation through existing datasets, and in simulation through use of the CARLA Simulator. In experimentation, metrics including detection precision, recall, and mean intersection over union are analyzed. In simulation, additional metrics are analyzed in-

cluding track position error and velocity error. Results for both experimentation and simulation indicated high precision (>98%) and slightly worse recall ($> 94\%$), which asserts that LiDAR sensors are indeed useful for object detection as other works have also found. In the tracking results for simulation, centroid position errors of $0.3-0.4m$ and high velocities errors of up to $6.9m/s$ are observed. Poor performance in these areas may be associated to noisy estimations of the object's centroid position in the detection process.

5.3 Conclusions

In conclusion, LiDAR sensors are observed to be a valid sensor choice that can be used for multi-object detection and tracking of pedestrians, vehicles, and other similarly sized objects. Because their stand-alone use may be limited in their detection range, they may serve as a complementary sensor to traditional sensors such as RADARs and or cameras. Their independence for illumination requirements and direct representation of 3D space can solve short-comings of cameras, and their increased resolution can allow for orientation detection of vehicles when compared to RADARS. Lastly, the need for other sensing modalities in addition to LiDAR can also be seen when partial scans of vehicles are obtained either due to sensor occlusion, weak returns from low reflective targets, and other possible factors. In Section 2.2, discussion on the preference for LiDARs that are configured to scan below its sensor mid-plane or horizon was introduced. Below horizon scanning LiDARs, such as the Ouster OS1-32, are more preferential in infrastructure like applications compared to LiDARs that also scan above the sensor's horizon that often incurs waste.

5.4 Future Work

Future work for this thesis can include further developed methods for each of the object detection and tracking stages shown earlier in Figure 2.1.

For **Background Estimation**, future work can include:

- Dynamic updates of the background estimates. For instance, background measurements can be periodically taken if there are few / no objects in scene;
- Threshold tuning for occupancy map filtering methods,.

For the **Clustering and Detection** processes, additional work can investigate:

- Additional clustering algorithms that can be efficiently used in 3D Space;
- Refined orientation detection algorithms, with tuned weights and additional cost parameters.

Lastly, for the **Tracking and Filtering** processes, additional work can be performed on:

- Updating the detection-to-track cost function to incorporate additional features;
- Better motion models for vehicles;
- Tracking and filtering over the object's bounding boxes.

REFERENCES

[1] D. Amir, W. Dayong, L. Minh, and S. Srikanth, "Building a smart work zone using roadside lidar," 2021.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[3] W. Commons, "Dbscan," 2011. file: DBSCAN-Illustration.svg.

[4] M. Sualeh and G.-W. Kim, "Dynamic multi-lidar based multiple object detection and tracking," *Sensors (Basel, Switzerland)*, vol. 19, p. 1474, Mar 2019.

[5] P. SEITZ, "Lidar stocks are mapping the road ahead for self-driving cars," Mar 2021.

[6] Wikipedia, "Lidar — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Lidar&oldid=1044135263`, 2021. [Online; accessed 14-September-2021].

[7] A. Börcs, B. Nagy, M. Baticz, and C. Benedek, *A Model-Based Approach for Fast Vehicle Detection in Continuously Streamed Urban LIDAR Point Clouds*. Computer Vision - ACCV 2014 Workshops, Springer International Publishing, Apr 2015.

[8] J. Zhang, W. Xiao, B. Coifman, and J. P. Mills, "Vehicle tracking and speed estimation from roadside lidar," vol. 13, p. 5597–5608, 2020.

[9] J. Zhao, *Exploring the fundamentals of using infrastructure-based LiDAR sensors to develop connected intersections*. PhD thesis, Jan 2020.

[10] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," 2016.

[11] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "Rangenet++: Fast and accurate lidar semantic segmentation,"

[12] Y. Wang, T. Shi, P. Yun, L. Tai, and M. Liu, "Pointseg: Real-time semantic segmentation based on 3d lidar point cloud," *arXiv preprint arXiv:1807.06288*, 2018.

[13] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "Polarnet: An improved grid representation for online lidar point clouds semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9601–9610, 2020.

[14] V. K. Bandaru, "Purdue e-pubs algorithms for lidar based traffic tracking: Development and demonstration," 2016.

[15] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017.

[16] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation*, pp. 1–4, IEEE, 2011.

[17] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[18] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

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

[20] Z. Zhang, J. Zheng, H. Xu, and X. Wang, "Vehicle detection and tracking in complex traffic circumstances with roadside lidar," vol. 2673, p. 62–71, Sep 2019.

[21] R. Jonker and T. Volgenant, "Improving the hungarian assignment algorithm," *Operations Research Letters*, vol. 5, no. 4, pp. 171–175, 1986.

[22] T. Fortmann, Y. Bar-Shalom, and M. Scheffe, "Sonar tracking of multiple targets using joint probabilistic data association," *IEEE journal of Oceanic Engineering*, vol. 8, no. 3, pp. 173–184, 1983.

[23] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.

[24] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[25] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," 2017.

[26] A. Krämmer, C. Schöller, D. Gulati, and A. Knoll, "Providentia-a large scale sensing system for the assistance of autonomous vehicles," in *Robotics: Science and Systems (RSS), Workshop on Scene and Situation Understanding for Autonomous Driving*, 2019.