Design of Dust-Filtering Algorithms for LiDAR Sensors in Off-Road Vehicles Using the AI and Non-AI Methods

by

Ali Afzalaghaeinaeini

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

August 2022

THESIS EXAMINATION INFORMATION

Submitted by: **Ali Afzalaghaeinaeini**

Master of Science in **Mechanical Engineering**

Thesis title: Design of Dust-Filtering Algorithms for LiDAR Sensors in Off-Road Vehicles Using the AI and Non-AI Methods.

An oral defense of this thesis took place on August 17, 2022 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Martin Agelin-Chaab
Research Supervisor	Jaho Seo
Examining Committee Member	Yuping He
Thesis Examiner	Jing Ren, Associate professor

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

The performance of Lidar sensors degrades in the presence of dust. These particles can impact sensor measurements and cause robot perception algorithms to misinterpret data. This thesis proposes two distinct dust filtering methods to address this issue. These methods utilize both AI and non-AI techniques. Specifically, we designed various dust filters including the Low-Intensity Dynamic Outlier Removal (LIDROR) using intensity and range information. In addition, we proposed a voxel-based classification method with multiple classifiers, such as Random Forest (RF), Support Vector Machine (SVM), and Deep Neural Network (DNN).

Two dust LiDAR datasets were collected and labeled for evaluation purposes. All proposed algorithms were implemented in the Robotic Operating System, allowing for the testing of these filters in real time. Using labeled data, a comprehensive comparison was made between these two methods. The proposed filters outperform conventional filters in terms of achieving dust removal without losing the surrounding data.

Keywords: LiDAR, dust filtering, dust classification, LIDROR

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Ali Afzalaghaeinaeini

STATEMENT OF CONTRIBUTIONS

Parts of this thesis has been published for publications in the following:

A. Afzalaghaeinaeini, J. Seo, D. Lee, and H. Lee., "Design of Dust-Filtering Algorithms for LiDAR Sensors Using Intensity and Range Information in Off-Road Vehicles," *Sensors 2022, Vol. 22, Page 4051*, vol. 22, no. 11, p. 4051, May 2022, doi: 10.3390/S22114051.

A. Afzalaghaeinaeini, J. Seo, D. Lee, and H. Lee, "Design of a LIOR-Based De-Dust Filter for LiDAR Sensors in Off-Road Vehicles," *Engineering Proceedings 2021, Vol. 10, Page 70*, vol. 10, no. 1, p. 70, Nov. 2021, doi: 10.3390/ECSA-8-11338.

F. Hanafi Sheikhha, A. Afzalaghaeinaeini, and J. Seo, "Collaborative Tracking Control Strategy for Autonomous Excavation of a Hydraulic Excavator," *Engineering Proceedings 2021, Vol. 10, Page 43*, vol. 10, no. 1, p. 43, Nov. 2021, doi: 10.3390/ECSA-8-11333.

ACKNOWLEDGEMENTS

I would like to express my great appreciation for my wife for her assistance which enabled me to focus solely on the research presented in this work. Without her, this work cannot be done.

In addition, my supervisor, Dr. Jaho Seo, has been incredibly supportive and kind throughout the duration of my thesis. I cannot complete this thesis without his supervision, as he is an exceptional mentor whose advice greatly facilitates my research in this work.

Finally, I would like to thank my lab mates Arif, Omid, and Tyler for their help during this work. Their friendship made this experience memorable.

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS AND SYMBOLS

ROS Robotic Operating System

SOR Statistical Outlier Removal Filter

ROR Radius Outlier Removal Filter

DROR Dynamic Radius Outlier Removal Filter

LIOR Low-Intensity Outlier Removal Filter

LIDROR Low-Intensity Dynamic Radius Outlier Removal Filter

SVM Support Vector Machine

RF Random Forest

DNN Deep Neural Network

CNN Convolutional Neural Network

SLAM Simultaneous Localization and Mapping

KNN K-Nearest Neighbors

Chapter 1

Introduction

1.1 Context of study

Many researchers have recently been working on creating autonomous systems for off-road vehicles in a variety of industrial areas, including construction and mining [1, 2]. This is because these vehicles are able to perform certain tasks with high efficiency and accuracy, particularly repetitive and tedious tasks. In addition, autonomy may be advantageous for preventing humans from working in harsh situations. In these situations, autonomous robots can perform the task instead of humans.

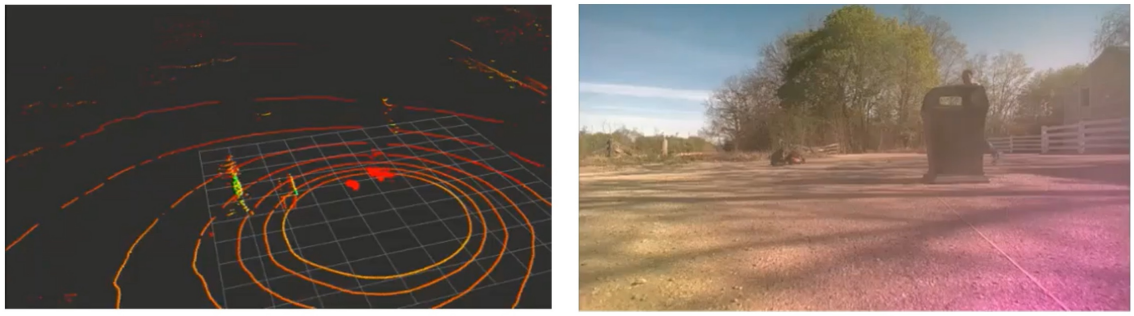
Autonomous operations consist of three components, including perception, planning, and control, as shown in Fig 1.1. The first is perception, in which robots use sensors such as Light Detection and Ranging (LiDAR) and depth cameras to sense

Figure 1.1: This figure shows different components of autonomous systems including perception, planning, and control.

their surroundings. The robot should then process these data and extract meaningful information for purposes such as object detection and simultaneous localization and mapping (SLAM) and finally make decisions, such as whether the robot should stop or continue moving, and send these commands to the robot's actuator.

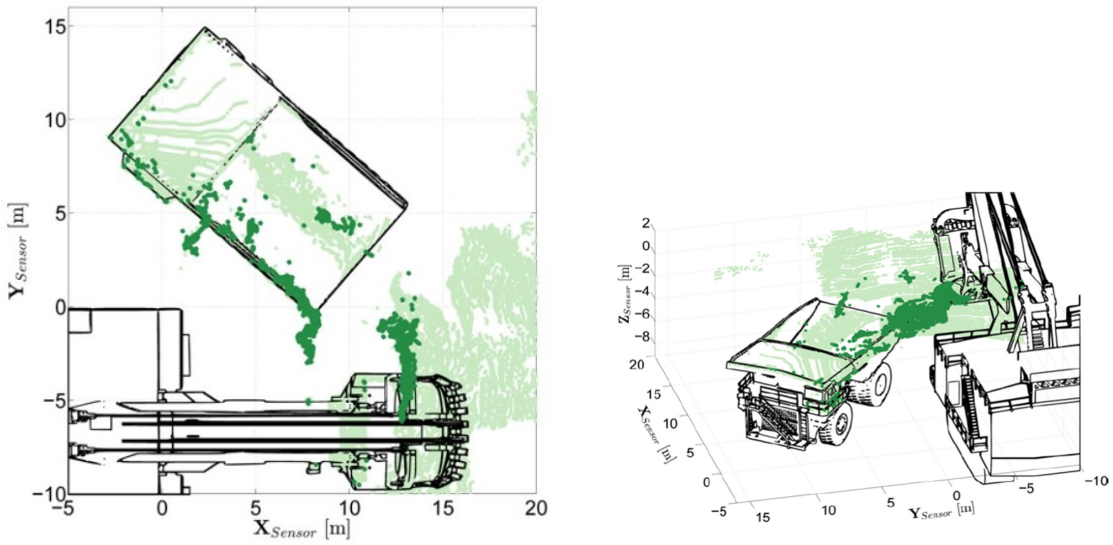
LiDAR is one of the most widely used sensors for performing perception in mobile robots because it can provide important information regarding the geometry and intensity of the surrounding environment. A popular LiDAR sensor, such as VLP-16 [4], can sense up to $600,000$ points per second in a distance of a hundred meters with an accuracy of three centimeters. The image and point cloud data generated by the VLP-16 sensor are shown in Fig. 1.2.

Due to the aforementioned benefits, the LiDAR sensor has a wide range of applications in mobile robotics, including object detection [5, 6], localization [7, 8], and mapping [9, 10]. However, measured data by the LiDAR sensor can be corrupted in the presence of adverse weather conditions such as dust [3, 11], snow [12, 13], and fog [14, 15].



(a) (b)

Figure 1.2: An example of a LiDAR point cloud (a) and its corresponding image captured by a depth camera (b).



(a) (b)

Figure 1.3: Effect of dust on LiDAR point cloud: top-view (a) and pictorial view (b) [3].

1.2 Effect of dust on 3-D LiDAR point cloud data

When exposed to extreme environmental conditions such as dust, the performance of LiDAR sensors is systematically degraded [3] because, unlike radar, most commercial LiDAR sensors operate at 900 nm wavelength, allowing them to detect airborne particles. For instance, Fig. 1.3 illustrates an example of a LiDAR point cloud in the presence of dust. As shown in Fig. 1.3b, the dust cloud resembles an object, which may cause the robots to misinterpret their surroundings. In such a scenario, LiDAR sensors may be unable to distinguish between data from dust clouds and data from non-dust clouds. During the competition, the winner of the DARPA urban challenge, Boss, had the same issue of misclassification of dust as an object. [16].

This behavior of the LiDAR sensor in adverse weather conditions could adversely impact the entire vehicle's perception. In order to use this sensor in these circumstances, it is necessary to identify and remove particles such as dust.

1.3 Research objective

The primary goal of this thesis is to present novel methods for detecting and removing dust particles using LiDAR sensors, which are commonly used in robotics, in order to improve the performance of robot perception in adverse weather conditions, especially dust. More specifically, the proposed methods target state-of-the-art technologies, including AI and non-AI techniques, to detect airborne particles in 3D LiDAR point cloud and minimize the effect of dust particles. The proposed solution expects to be experimentally evaluated using datasets collected. Finally, the developed algorithms will be implemented in Robotic Operating System (ROS) so that they can be utilized in real-time.

The detailed objectives of this research include:

1. Development of de-dust filtering algorithms based on AI techniques that can remove dust point clouds from a LiDAR sensor in dusty environments while keeping non-dust point clouds.

2. Development of de-dust filtering algorithms using methods that do not rely on artificial intelligence and are capable of minimizing the impact of dust.

3. Real-time implementation of the developed de-dusting filtering algorithms using the ROS.

4. Evaluation the performance of the proposed algorithms including both AI and non-AI techniques, in terms of evaluation metrics using the labeled data gathered in the presence of dust.

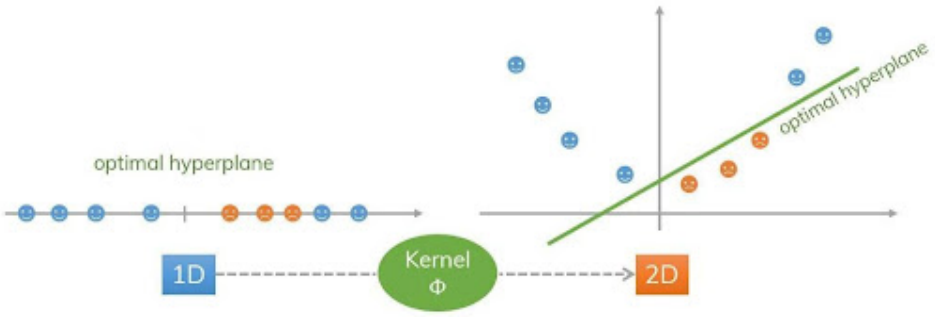


Figure 1.4: Separating the data by mapping it to a higher dimension using a polynomial kernel [17].

1.4 Working foundation

The theoretical foundations for the research presented in this thesis are discussed in the following sections. This includes an overview of the mathematics that underlies the techniques that were developed here. Furthermore, the ROS is described to provide the reader with a better understanding of how the proposed methods are integrated with ROS.

1.4.1 Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a robust and flexible Machine Learning model that can perform linear or nonlinear classification and regression [17, 18]. SVMs are based on the principle of projecting training data onto a higher-dimensional plane using the kernel concept to make the data linearly separable and then applying a large margin classifier to it there. There are numerous techniques for projecting data into higher dimensions, one of which is illustrated in Fig. 1.4 named polynomial kernel [17].

There are two types of large margin classification: hard margin classification and soft margin classification. As shown in Fig. 1.5, the hard margin classifier requires

all instances be off the street and on the right side. Several issues exist with hard margin strategy. First, it is applicable only when the data can be separated linearly. Second, it is sensitive to outliers, which makes this classifier unsuitable for real data containing noise. SVM with a hard margin is defined in Equations (1.1) and (1.2).

$$f(w,x) = \begin{cases} 0, & w^T x + b \ < 0 \\ 1, & w^T x + b \ > 0 \end{cases} \tag{1.1}$$

$$\begin{gathered} minimize_{w,b} \, (\frac{1}{2} w^T w) \text{ subject to } t^i (w^t x + b) \geq 1 \\ \text{for } i = 1, 2, ..., m \end{gathered} \tag{1.2}$$

where w and b are hyperplane coefficients found after solving SVM and x are the input features to the classifier for prediction, and m is the number of training points. This strategy maximizes the width of the street between the two classes.

Soft margin classification is a more flexible model that can be used to avoid aforementioned two issues. The objective of this method is to find a balance between maximizing street width and minimizing margin violations (i.e., instances that end up in the middle of the street or even on the wrong side). Soft margin is represented by Equation (1.3).

$$\begin{gathered} minimize_{w,b,\zeta} \, (\frac{1}{2} w^T w + C \sum_{i=1}^{m} \zeta^i) \text{ subject to } t^i (w^T x + b) \geq (1 - \zeta^i) \\ \text{for } i = 1, 2, ..., m \text{ and } \zeta^i \geq 0 \end{gathered} \tag{1.3}$$

where w, b, x, and m are identical to hard margin classifier while for each instance, the slack variable ζ^i with coefficient C is added to a hard margin indicating how much the i^{th} instance is permitted to deviate from the margin. As shown in Fig. 1.6, the higher the value for C, the less likely a violation will occur, and vice versa. Two hyperpa-

rameters affect the classification performance of SVM. The first hyperparameter is selection of the Kernel for data separation. The choice of Kernel consist of polynomial, Gaussian RBF, and Sigmoid. The second hyperparameter is the constant C, which determines the margin violation tolerance. The selection of appropriate parameter values is crucial for the training of this algorithm.

1.4.2 Random Forest (RF)

A random forest is a collection of Decision Trees trained with the bagging technique. The number of estimators indicates the number of decision trees utilized by Random Forest. Scikit-Learn uses the classification and regression tree (CART) al-

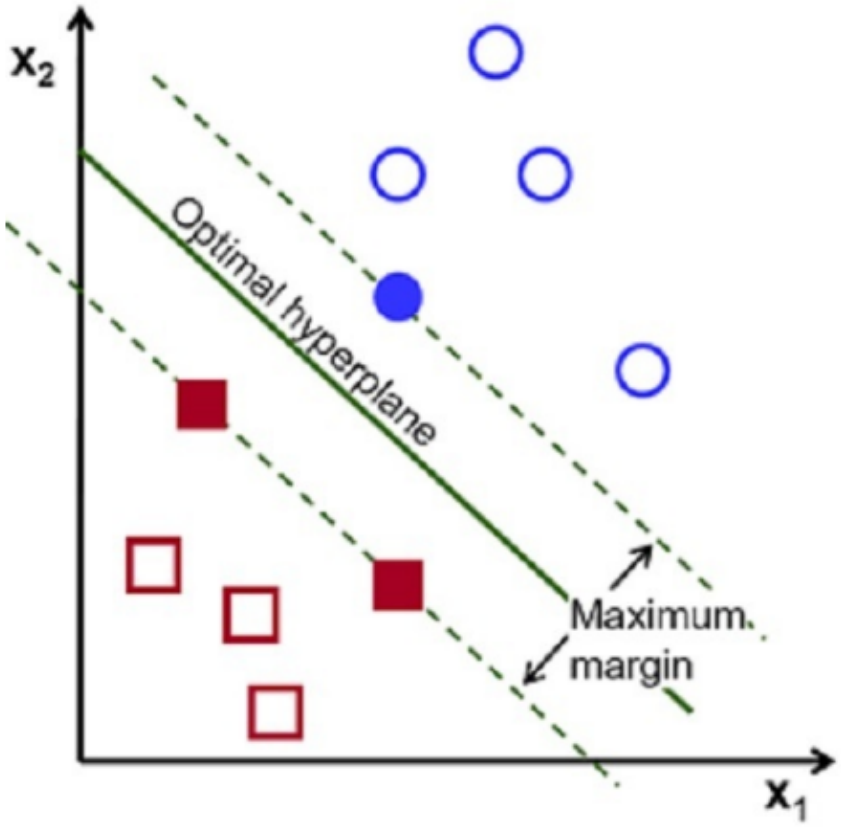


Figure 1.5: Large margin classification illustration [19].

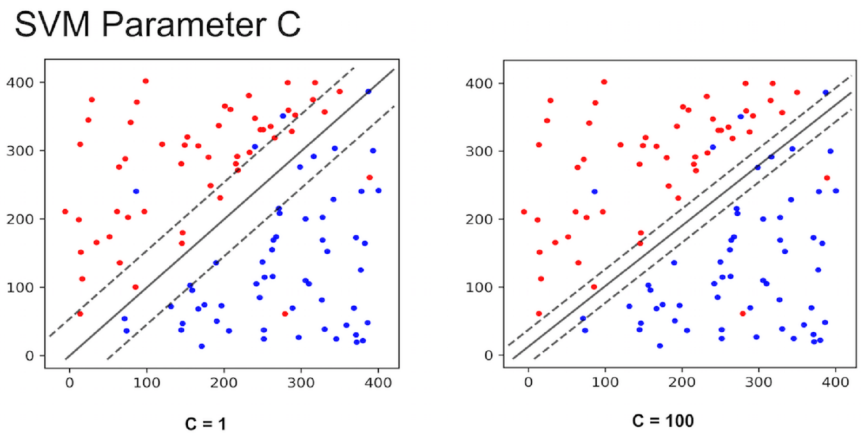


Figure 1.6: The effect of coefficient C on the soft margin classification decision boundary [18].

gorithm to train Decision Trees. Using a single feature k and a threshold t_k, the algorithm initially divides the training set into two subsets. It looks for the pair (k, t_k) that produces the purest subsets.

The cost function that this algorithm attempts to minimize is given in Equation (1.4):

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{1.4}$$

where G_{left} and G_{right} represent the impurity of the left and right subset and m_{left} and m_{right} represents the number of instances in the left and right subset. The Gini impurity criteria is defined in equation (1.5):

$$G_i = 1 - \sum_{k=1}^{n} P_{i,k}^2 \tag{1.5}$$

where $P_{i,k}$ is the class k instance ratio among the training instances in the i^{th} node. After successfully splitting the training set into two, the CART algorithm divides the subsets using the same logic, then the next subsets, and so on, recursively. When it reaches the maximum depth, it stops recursing.

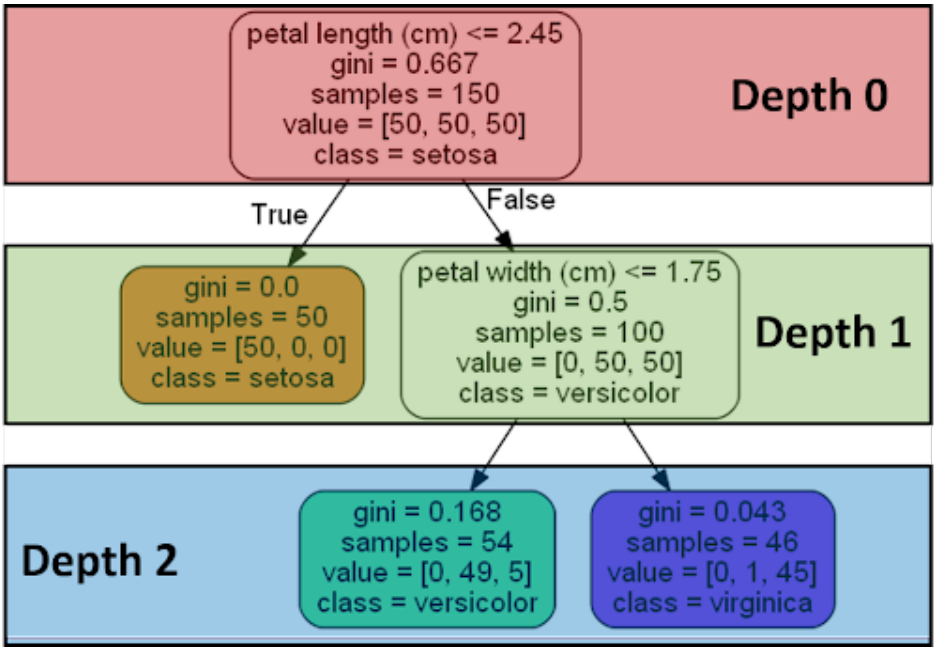


Figure 1.7: Example of a decision tree classifier [20].

The maximum depth, minimum samples split, and minimum samples leaf are some Decision Tree hyperparameters that can be tuned to find a better solution. The Fig. 1.7 depicts an example of a Decision Tree Classifier. In this example, the classifier has a depth of three.

1.4.3 Principal Component Analysis (PCA)

One of the most widely used dimensionality reduction strategies is Principal Component Analysis (PCA). This algorithm finds the hyperplane that is closest to the data and projects it onto it. Fig. 1.8 illustrates this. PCA determines which axis accounts for the most variance in the training set. It also identifies the second orthogonal axis that accounts for the greatest amount of residual variance, and so on.

Using PCA, it is possible to calculate the eigenvalues and eigenvectors of the point cloud. Fig 1.9 depicts an illustration of eigenvalues and eigenvectors for a simple example of a groups of LiDAR points located inside a cube. When all of the points are

roughly arranged in two dimensions, the eigenvalue in the last dimension is negligible in comparison to the other eigenvalues. It can be inferred that the axis with the most points aligned in its direction has a higher eigenvalue. This method was used to extract geometrical information from the LiDAR point cloud for training AI models, as explained in Chapter 3. An example of this information is the planarity, that is, whether the points are scattered or roughly aligned.

1.4.4 Deep Neural Network (DNN)

The neural network is based on biological neurons. [23] developed a fairly simple model of the biological neuron: it includes one or more binary (on/off) inputs and one binary output. When more than a specific number of its inputs are active, the artificial neuron activates its output. In their study, they demonstrated that even with such a simple model, it is feasible to construct a network of artificial neurons capable of computing any logical assertion.

Following that, [24] created the Perceptron. It is built on a slightly modified artificial neuron known as a threshold logic unit (TLU). The inputs and outputs are integers, and each input connection has a weight assigned to it. The TLU computes a weighted sum of its inputs, then applies a step function to that amount to get the re-

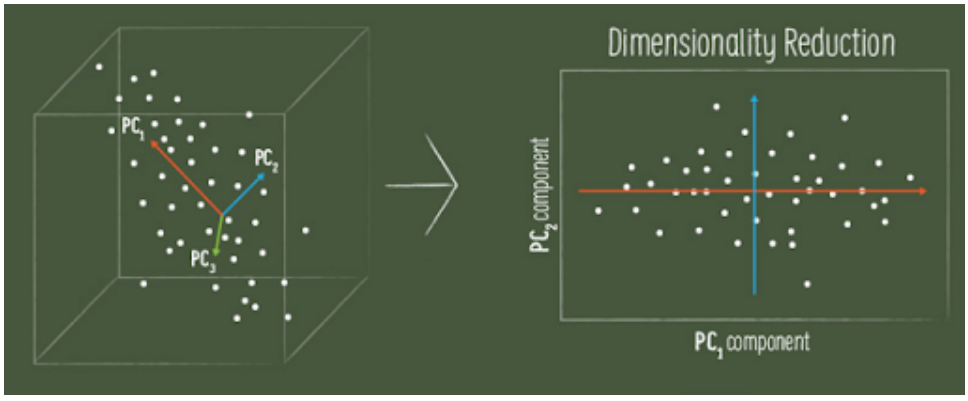


Figure 1.8: A dimensionality reduction example using PCA [21].

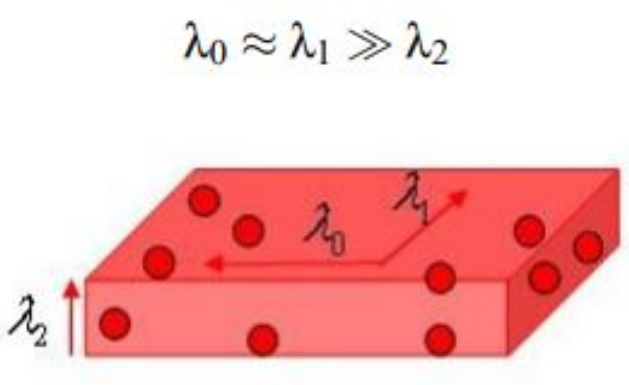


Figure 1.9: An illustration of the concept of eigenvalues in point cloud data [22].

sult. For simple linear binary classification, a single TLU can be utilized. It takes the inputs and computes a linear combination of them. A layer is called a fully connected layer or a dense layer when all of the neurons in it are connected to every neuron in the preceding layer.

$$z = w_1 x_1 + w_2 x_2 + ... + w_n x_n = w^T x$$
$$hevisise(z) = \begin{cases} 0, & \text{if z } < 0 \\ 1, & \text{if z } > 0 \end{cases} \tag{1.6}$$

where w_i are coefficients corresponding to input x_i, and z is a linear combination of the input x_i.

The inputs of the Perceptron are fed by unique passthrough neurons called input neurons, which output whatever input they receive. Furthermore, an additional bias component is usually added: it is normally represented using a unique sort of neuron known as a bias neuron.

$$h_{w,b}(X) = \phi(W^T X + b) \tag{1.7}$$

where X represents the matrix of input features, W contains all the connection weights

except the one from the bias neuron, and b contains all the connection weights between the bias neuron and the artificial neuron. The function is called the activation function: when the artificial neurons are TLUs, it is a step function.

The perceptron learning rule promotes connections that assist reduce mistakes. More exactly, the Perceptron receives one training instance at a time and makes predictions for each. It reinforces the connection weights from the inputs that would have contributed to the right prediction for each output neuron that generated an erroneous prediction. The rule is demonstrated using the equation (1.8).

$$W_{i,j}^{\text{next step}} = W_{i,j} + \alpha(y_j - \bar{y_j})x_i \tag{1.8}$$

where $W_{i,j}$ is the connection weight between the i^{th} input neuron and j^{th} output neuron, x^i is the i^{th} input value of the current training instance, y_j is the output of the i^{th} output neuron for the current training instance, $\bar{y_j}$ is the target output neuron for the current training instance, and α is the learning rate.

Because each output perceptron's decision boundary is linear, perceptrons are incapable of learning complicated patterns. If the training cases are linearly separable, however, this technique will converge to a solution.

Some of perceptron's drawbacks can be overcome by stacking several perceptrons, a technique known as Multilayer Perceptron (MLP). An MLP is made up of one input layer, one or more hidden levels of TLUs, and one final layer of TLUs termed the output layer. The parameters that should be tuned for each problem are the number of hidden layers and the number of nodes in each layer. Except for the output layer, each layer contains a bias neuron that is entirely connected to the following layer. The approach is then trained with a backpropagation algorithm. This algorithm is made up of numerous steps:

1. It works with one mini-batch at a time (for example, 32 instances each), and it passes through the entire training set numerous times. Each pass is referred to as an epoch.

2. Each mini-batch is sent to the network's input layer, which then forwards it to the first hidden layer. After that, the algorithm computes the output of all the neurons in this layer (for every instance in the mini-batch). The result is sent on to the next layer, whose output is computed and passed on to the next layer, and so on until we reach the last layer, whose output is computed and passed on to the next layer. This is the forward pass: it is the same as creating predictions, except that all intermediate findings are saved because they are required for the backward pass.

3. The algorithm then calculates the network's output error. To do so, it employs a loss function that compares the desired output to the network's actual output and gives the error value.

4. The error is then calculated based on how much each output connection contributes to the error. This is accomplished analytically by employing the chain rule.

5. The method then uses the chain rule to determine how much of these mistake contributions occurred from each connection in the layer below, going backward until the algorithm reaches the input layer. As previously stated, this reverse pass network propagates the error gradient backward across the network.

6. Finally, the algorithm uses the error gradients it just generated in a Gradient Decent step to modify all of the network's connection weights.

To make this technique work, the authors in [25] made a significant improvement to the MLP's architecture: they replaced the step function with a logistic function, allowing gradient descent to make some progress at each step. Equation (1.9) defines the logistic function.

$$\sigma(z) = \frac{1}{1 + exp(-z)} \tag{1.9}$$

The sigmoid function, however, is not the only option for an activation function. Other activation functions include the Rectified Linear Unit (ReLU) and Hyperbolic Tangent function (Tanh). These alternatives, as well as their mathematical representation, are shown in Equations (1.10) and (1.4.4), respectively.

$$Tanh(z) = \frac{exp(z) - exp(-z)}{exp(z) + exp(-z)} \tag{1.10}$$

$$ReLU(z) = \begin{cases} z, & z > 0, \\ 0, & z < 0, \end{cases} \tag{1.11}$$

Choosing the loss function may differ depending on the application. The cross-entropy loss function is commonly applied in binary classification, as seen in Equation (1.12).

$$\text{cross entropy loss function} = \frac{1}{N} \sum_{i=1}^{n} -\big(y_i \log(\bar{y}) + (1 - y_i) \times \log(1 - \bar{y})\big) \tag{1.12}$$

where N is the number of the training set, y_i is a true label, and $\bar{y}$ is a predicted value.

To compute the gradient descent, several techniques such as ADAM [26], NADAM [27], and momentum can be used. Depending on the application, each of these algo-

rithms can be tested to see which one has the best performance for the situation at hand. The table below summarises the list of Deep Neural Network parameters that may require adjusting.

Table 1.1: Deep Neural Network Parameters

Parameters	value
No. Hidden layers	Usually between 2 and 5
Number of nodes in each layer	No specific range
Learning rate	Between 10^{-5} and 10^{-1}
Optimization technique	ADAM, NADAM, momentum

1.4.5 Robotic Operating System (ROS)

The Robot Operating System (ROS) is a flexible framework that includes numerous tools and libraries for the development of robotic applications. It has various advanced capabilities that aid developers in tasks such as message forwarding, distributed computing, code reuse, and designing cutting-edge algorithms for robotic applications. Morgan Quigley founded the ROS project in 2007, and it was maintained at Willow Garage, a robotics research facility for creating hardware and open-source software for robots. ROS's purpose was to create a standard means of programming robots while also providing off-the-shelf software components that could be readily combined with custom robotic applications.

There are numerous reasons to select ROS as a framework for programming. First, the ROS ecosystem is equipped with tools for debugging, visualization, and simulation. For example, RViz is one of the most effective open-source visualization tools. ROS also permits the use of a variety of device drivers and interface packages for a wide range of robotic sensors and actuators. Among the high-end sensors are 3D LIDAR, laser scanners, depth sensors, and actuators. These components can be ef-

fortlessly connected to ROS.

Furthermore, the ROS message-passing middleware facilitates communication between a variety of programs. This middleware is referred to as nodes in ROS. These nodes are writable in any language supported by ROS client libraries. C++ and Python are two examples of these programming languages. ROS is one of the frameworks that is used the most frequently in the robotics industry due to the reasons that were discussed earlier.

1.5 Thesis outline

The rest of the thesis is organized into four sections. The second chapter is a review of previous efforts to overcome the challenge of autonomous robot perception using LiDAR sensors in adverse weather conditions such as dust. Chapter 3 proposes the methodology for developing dust-filtering algorithms for 3-D LiDAR point clouds. The results are then analyzed and discussed in chapter 4. Chapter 5 ends this thesis with a summary of the work, a discussion of the findings, and suggestions for future research.

Chapter 2

Literature Review

This chapter provides the information necessary to better realize the work presented in this thesis. First Section 2.1 discusses the related work regarding how dust particles affect the LiDAR measurements. Section 2.1, and 2.2.4 introduce the previous studies that have tried to minimize the adverse effects of averse weather conditions using techniques such as non-AI filtering methods and AI techniques. Then Section 2.4 presents the existing open data sets that are available online. Finally, this chapter ends with Section 2.5, highlighting the research gap.

2.1 Effect of dust on LiDAR data

The influence of dust particles on LiDAR data was thoroughly investigated by [3]. In this paper, the authors classify four distinct dust particle behaviors that can be detected by a LiDAR sensor. These behaviors are shown in Fig. 2.1. In scenario (a), when the dust cloud is sparse, the LiDAR beam is able to penetrate it, and the dust cloud does not produce noise in LiDAR measurements. In scenario (b), when the dust cloud is dense, LiDAR images the surface of the dust cloud. The third

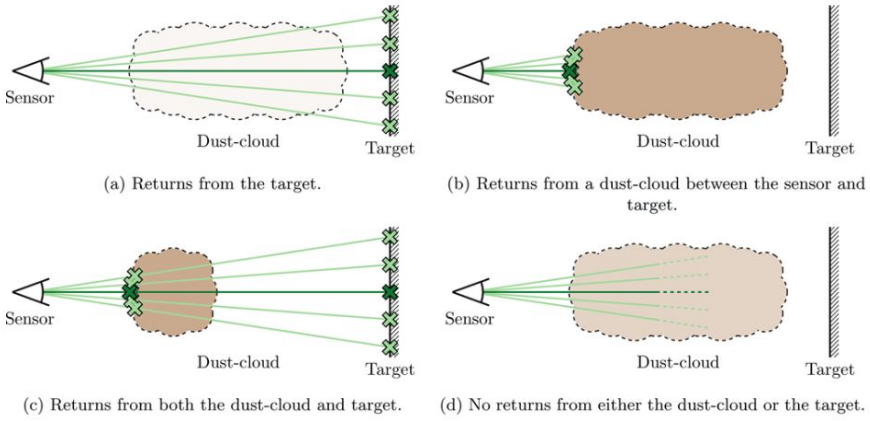


Figure 2.1: Four behavior of LiDAR sensor in the presence of dust [3].

scenario is a transition between the first two. There are also exceptional situations in which no LiDAR beam can be returned from an object corresponding to scenario (d). They also studied the dust cloud parameters that influence the LiDAR return. As shown in Fig. 2.2, these parameters include the distance between a LiDAR and dust clouds, the distance between a LiDAR and a target, the dust cloud's length, the dust density, the dust particle's size, and the reflectivity and the surface area of a target (reflected points). In this thesis, these parameters were used when LiDAR data was to be collected.

Additionally, they examined the impact of different dust cloud thicknesses at various sensor distances. In mining applications, they also provided qualitative results for dust-impacted LiDAR point clouds. They showed that dust particles have a systematic effect on LiDAR measurements. Dust, for instance, affects the return intensity of other targets behind it. Fig. 2.3 presents various conditions corresponding to the situation in Fig. 2.1 when the dust is present. In case (b), for example, dust prevented the collection of LiDAR data from the target.

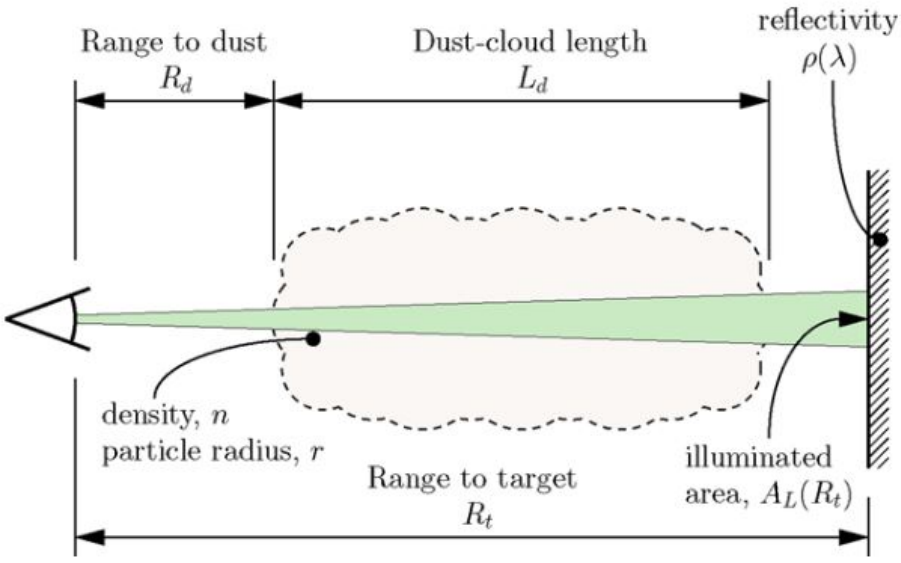


Figure 2.2: Parameters of dust cloud that impacts the LiDAR return [3].

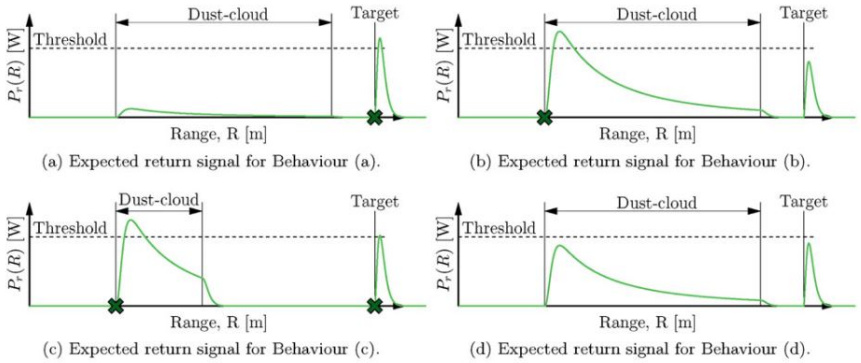


Figure 2.3: LiDAR return for the cases corresponding to 2.2 [3].

2.2 Non-AI-based techniques for noise removal in adverse weather conditions

In this part, a variety of non-AI de-noise filters for LiDAR point clouds that have been utilized to improve the quality of detection in severe weather conditions are reviewed. These filters include Statistical Outlier Removal Filter (SOR), Radius Outlier Removal Filter (ROR), Dynamic Outlier Removal Filter (DROR), and lastly, Low-Intensity Outlier Removal Filter (LIOR).

2.2.1 SOR filter

The objective of the SOR filter is to eliminate sparse outliers resulting from measurement error [13, 28]. To accomplish this, the algorithm iterates over each point and calculates the average distances d_i of k-nearest points to that point, where k is an integer parameter of the filter that can be chosen based on how many neighbor points are to be analyzed [28]. As another important variable, the threshold value T can be defined as follows in Equation (2.1).

$$T = \mu \pm \beta \times \sigma \tag{2.1}$$

where μ and σ are the mean and standard deviation of the average distances d_i, and β is a constant multiplier. This filter removes any points whose average distances exceed the threshold interval. The effectiveness of the SOR filter is dependent on the selection of β and k. The authors of [13] utilized this filter as a potential filter for de-snowing.

2.2.2 ROR filter

The ROR filter [13, 29] eliminates isolated outliers from point clouds by iteratively traversing each point and counting the number of points fall inside a sphere with the point's center and search radius, R. The filtering process is depicted in Fig. 2.4. It utilizes the k- d tree algorithm [30] to find a point within a sphere. If the number of points is less than the minimum acceptable number of points N, it is discarded as an outlier, otherwise saved as an inlier. The parameters N and R can be used to determine the optimal ROR filtering solution.

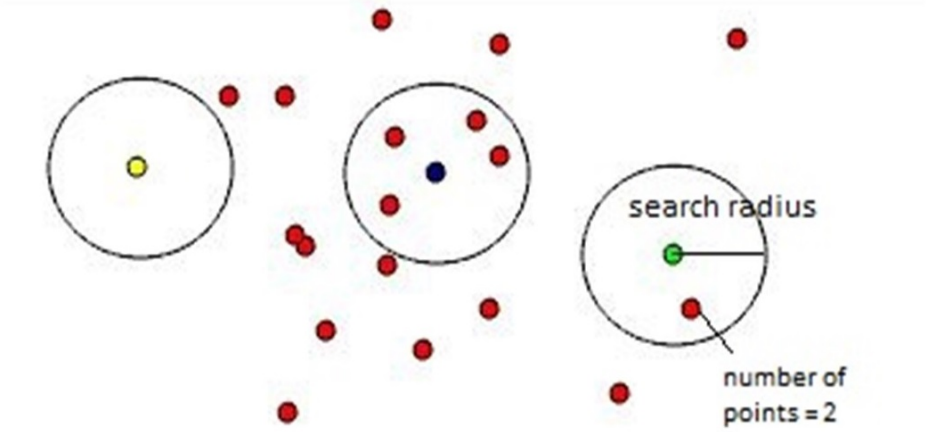


Figure 2.4: Illustration of how the ROR filter works [29].

2.2.3 DROR filter

In [13], the ROR and SOR filters were selected to evaluate their de-snowing performance. This study figured out that the SOR was able to remove the majority of snow points but was unable to remove snow points that were densely clustered. In addition, although the ROR filter showed superior performance for de-snowing in general, it eliminates all relevant environmental information that was farther than 18 meters from a LiDAR sensor. This is due to the reason that LiDAR point clouds become sparser as sensor distance increases, whereas the search radius of the ROR filter remains constant.

In order to address this issue, the research developed a DROR filter in which the search radius varies proportionally with distance from the LiDAR sensor, as shown in Equation (2.2).

$$R_{dynamic} = \phi \times \alpha \times \sqrt[2]{x^2 + y^2} \tag{2.2}$$

where ϕ is a constant multiplier, α is the LiDAR sensor's angular resolution, and $[x, y]$ are the Cartesian coordinates of the point. In Fig. 2.5, the pseudo-code for this filter is

Algorithm 1 DROR filter

FOR (Each point in the point cloud)
    Search radius $\leftarrow \sqrt{x_p^2 + y_p^2}$
    IF (search radius $<$ minimus search radius)
        search radius = minimum search radius
    ELSE
        Search radius $\leftarrow \phi \times \alpha \times \sqrt{x_p^2 + y_p^2}$
    ENDIF
        $n \leftarrow$ Find number of points inside search radius
        IF ($n <$ threshold point)
            Outliers $\leftarrow$ point
        ELSE
            Inliers $\leftarrow$ point
    ENDIF
ENDFOR

Figure 2.5: Pseudocode of the DROR filter.

shown . The dynamic radius in Equation (2.2) allows rich data from the surrounding environment to be preserved while snow particles are removed. To avoid a very small search radius for points close to the LiDAR sensor, search radii smaller than the minimum search radius were set equal to the minimum search radius in the study.

2.2.4 LIOR filter

For denoising, the methods described above rely solely on geometry information from a LiDAR sensor. Based on the observation that snow particles have a lower intensity value than other objects, [31] utilized the intensity information from LiDAR's 3D point clouds for de-snow filtering. The study [31] proposed the two-stage LIOR filter by applying this principle. The initial stage consists of iterating through each point and identifying those whose intensity is less than a threshold intensity value ϵ. Choosing the proper threshold is essential for the proper operation of the LIOR filter.

In the second step, the ROR filter was applied to the candidate outliers that had been identified in the first phase. All parameters associated with the ROR filter,

Algorithm 2 LIOR filter

```
FOR (Each point in the point cloud)
    IF (point intensity > threshold intensity)
        Inliers ← point
    ELSE
        % SR is sesrch radius
        n ← Find number of points inside SR
        IF (n < threshold point)
            Outliers ← point
        ELSE
            Inliers ← point
        ENDIF
    ENDIF
ENDFOR
```

Figure 2.6: Pseudocode of the LIOR filter.

including the minimum acceptable number of points and the search radius, play a crucial role at this stage. Those points identified as outliers in the second step were subsequently eliminated from the point cloud.

The preceding procedure is depicted in Fig. 2.6. The main feature of this filter is to apply the ROR only to selected points. This enables the LIOR filter to achieve a higher speed than the DROR filter while maintaining the same high level of snow-removal performance as the DROR filter [31].

2.3 AI-based techniques for noise removal in adverse weather conditions

The AI-based techniques in the literature for the perception of LiDAR sensors can be divided into two broad categories: Deep learning methods and other machine learning methods such as Support Vector Machine (SVM) and Random Forest (RF). In general, deep learning frameworks for the LiDAR point cloud can be classified into four types [32]:

1. Models based on voxels, such as VoxNet [33] and 3D ShapeNet [34]

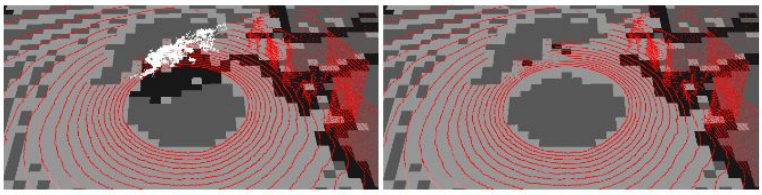


Figure 2.7: Results of voxel-based classification of dust particles [11].

2. Models based on point clouds, such as PointNet [35] and PointNet++ [36]

3. Models based on graphs, such as SyncSpecCNN [37] and Edge-Conditioned Convolution[38]

4. Models based on views, such as MultiViewCNN [39] and MVCNN-MultiRes [40]

Each type of the mentioned classifiers has a unique set of input data. For voxel-based models, the point cloud is first converted into voxels, which are then fed into the models. The input for the AI architecture for a point-based model is a point. The graph-based model first converts the point cloud data into a graph data structure and then feeds it to the neural network, while the view-based model converts the point cloud data into images and feeds them to the AI model.

The AI-based techniques can also be classified into 3D point cloud semantic segmentation [41], 3D object detection [42, 43], and 3D object classification [44, 45] based on their applications. Some deep learning techniques have been modified to be effective for dust classification. The authors of [11], for example, created a point-based and voxel-based deep learning architecture for distinguishing dust particles from other point clouds. The result of this work is shown in Fig. 2.7. These techniques have also been used to combat other adverse conditions such as fog and snow. [12] removed snow from the LiDAR point cloud using CNN methods.

Other machine learning techniques have also contributed significantly to this field. For example, in [22], the authors used geometrical features obtained from a Principal

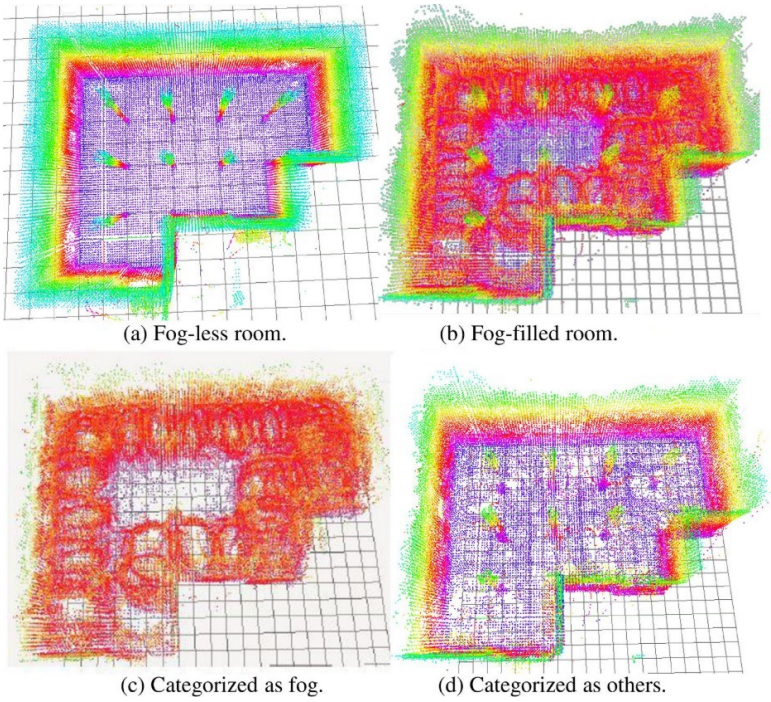


Figure 2.8: Results of classification of fog using SVM and KNN [14].

Component Analysis (PCA) to perform ground classification using a Gaussian Mixture Model (GMM). [46] proposed using LiDAR intensity returns as a feature in a Support Vector Machine (SVM) classifier to recognize vegetation from road surfaces. [47] used a Random Forest (RF) classifier to classify up to four terrain types using both geometrical features and intensity returns. These techniques, such as deep learning methods, have also been applied to other adverse conditions such as fog and snow. For example, in [14], the author used both SVM and KNN methods for defogging and concluded that the SVM performed better. The result of this work is shown in Fig. 2.8.

The LiDAR sensor can provide two primary data sources that have been extensively utilized by AI classifiers as input features. The first type of data is geometry data, which is returned by the LiDAR sensor and enables the calculation of the cartesian coordinates of surrounding points. The second type of information is intensity

information, which provides information regarding the data's intensity. These data can be then analyzed to obtain meaningful information about the environment.

2.3.1 Geometry features

There are some features that have been used in the literature for the voxel-based classification technique showing good results. The examples of these features are slope [48], roughness [48], curvature [32], local linearity [32], and local planarity [32]. These features can be computed by applying the principal component analysis (PCA) to the voxels and computing the eigenvalues, as seen in equations (2.3) - (2.7) can be used.

$$\text{slope} = arcsin(\frac{\gamma_3}{\gamma_1}) \tag{2.3}$$

$$\text{roughness} = \gamma_3 \tag{2.4}$$

$$\text{curvature} = \frac{\gamma_1}{\gamma_1 + \gamma_2 + \gamma_3} \tag{2.5}$$

$$\text{linearity} = \frac{\gamma_1 - \gamma_2}{\gamma_1} \tag{2.6}$$

$$\text{planarity} = \frac{\gamma_2 - \gamma_3}{\gamma_1} \tag{2.7}$$

where γ_1,γ_2, and γ_3 are eigenvalues.

2.3.2 Intensity features

A prestigious LiDAR sensor such as Velodyne provides both intensity and geometry data. Intensity value represents the intensity of light received by the sensor from the emitted beam. Some researches have been conducted to identify the factors that

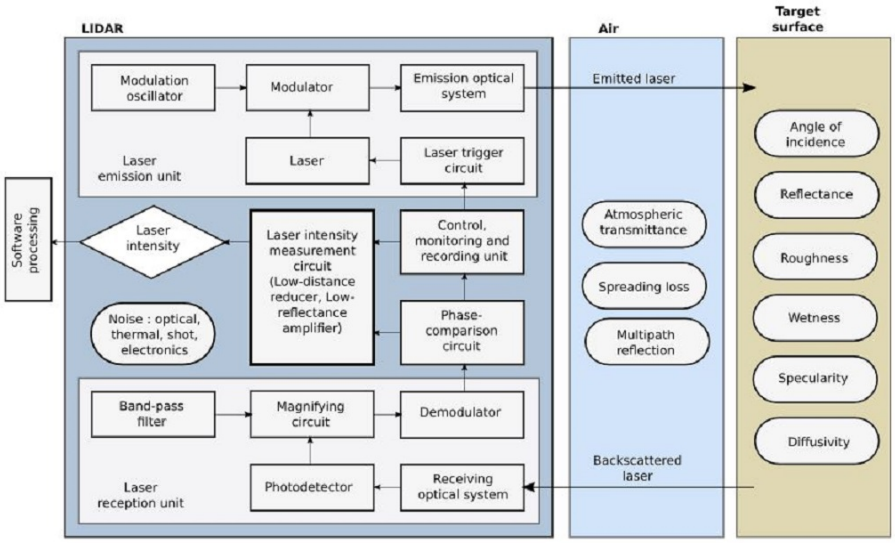


Figure 2.9: Different parameters that can affect the final value reported by LiDAR sensor as intensity [49].

can influence this value. For example, the authors of [49] figured out that intensity value is affected by three major factors: the internal processes of a sensor, the medium through which the LiDAR beam is emitted, and the target surface. The detailed list of these parameters is displayed in Fig. 2.9.

Some studies, including [11, 48], have used the fact that each object's surface can have its own intensity value due to its unique reflectance as a feature for training AI algorithms. Examples of a intensity-based feature are the mean intensity and the standard deviation intensity of the points located within a voxel.

2.4 Existing LiDAR dataset

Several datasets containing LiDAR data are available to public, including the well-known KITTI dataset [50], the A*3D Dataset [51], the nuScence Dataset [52], the Oxford RobotCar Dataset [53], the Canadian Adverse Driving Conditions Dataset [54], and the Waymo open dataset [55], but none of them contain dust datasets. The

Marulan dataset [56] includes LiDAR data with airborne particles such as dust and smoke. However, no information regarding intensity is provided here, and only 2D LiDAR sensors were utilized in their experiments. In order to develop the proposed dust filtering algorithms using a 3D LiDAR, it was necessary to generate new datasets containing dust.

2.5 Research gap

This thesis attempts to fill in a few of the gaps that are present in the previously published literature. The following provides a summary of these limitations:

1. Many researchers have developed autonomous off-road vehicles. These vehicles operate in extreme weather, which includes dust. Surprisingly few studies on dust filtering in adverse weather conditions have been published. Additionally, current literature solely relies on deep learning methods in this area. However, these AI techniques have the inherent problem of requiring a large number of data sets, resulting in high computation costs and training time.

2. Prior research has primarily focused on AI methods or non-AI methods. None of them offer a comprehensive comparison of these techniques.

2.6 Research contribution

The contribution of this research can be summarized as follows:

- To the best of our knowledge, the proposed non-AI solutions are the first attempt to create dust-filtering algorithms using non-AI techniques that take advantage of the inherent characteristics (intensity value) of dust point-cloud data.

- The proposed non-AI method can overcome the inherent problems of AI methods used for dust filtering, which require a large number of data sets for training, resulting in high computation costs and training time.
- This study considers a dataset with various scenarios in the presence of dust.
- This study provides an in-depth and comprehensive discussion of various design methodologies including both AI and non-AI techniques. Therefore, it can offer practical recommendations on which is the most suitable method through a comparative analysis.

Chapter 3

Methodology

The LiDAR sensor's performance is degraded when exposed to adverse environmental conditions such as dust [3]. They operate at 900 nm wavelength, allowing them to detect airborne particles. In such a case, LiDAR sensors may be unable to distinguish between data from dust clouds and data from non-dust clouds, reducing efficiency and robustness of the robot's perception.

To address the aforementioned issues, this chapter presents developed filtering technique to filter out dust particles from the LiDAR point cloud. These techniques can be divided into two main categories, including AI-based methods and non-AI-based methods. Then these proposed solutions were tested experimentally using two different datasets representing different outdoor scenarios. Following that, the dataset was manually labeled based on prior knowledge of the experimental environment. We evaluated the performance of the designed filters using the labeled dataset.

The rest of this chapter is organized into four sections. Section 3.1 and 3.2 describe the proposed non-AI techniques and AI techniques used for dust removal, respectively. Section 3.3 explains how we created the datasets and the ground truth labeling methods. Finally, Section 3.4 discusses the design of a mobile platform necessary for testing

the developed filtering algorithms in real-time.

3.1 Design of non-AI dust-filtering algorithms

One of the major goals is to create filtering algorithms capable of detecting and removing dust from a LiDAR point cloud. To accomplish this, some of the filtering methods that have shown promising results in other adverse conditions, such as snow, were selected to design dust filters. Statistical Outlier Removal Filter (SOR), Radius Outlier Removal Filter (ROR), Dynamic Radius Outlier Removal Filter (DROR), and Low-Intensity Outlier Removal Filter (LIOR) are examples of these algorithms. These filters were then designed to adapt to dust conditions. The general procedure for designing these filters is shown in Fig. 3.1. The filters' inputs, as shown in the figure, are raw LiDAR point clouds with geometry and intensity information. These filters' outputs are dust-free point clouds. In Fig. 3.1, each filter has some parameters that influence the filter performance. The filter parameters were then optimized based on the feedback provided by the chosen evaluation metrics. The filter parameters are then optimized based on the feedback provided by the chosen evaluation metrics. The final parameters of these filters are presented in Table. 3.1.

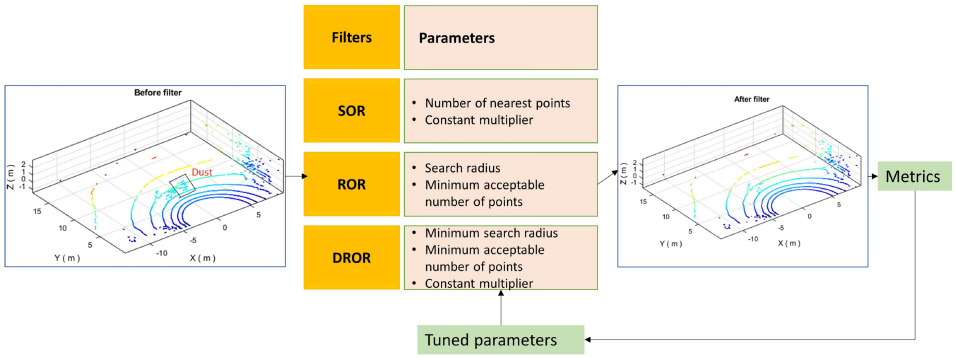


Figure 3.1: An illustration of the designing process of non-AI techniques.

Table 3.1: Selected conventional filters' final parameters

Filters	Parameters
SOR	No. of nearest points = 3 Multiplier constant = 0.1
ROR	Search radius = 0.04 (m) Minimum acceptable No. of points = 3
DROR	Constant multiplier = 0.008 Minimum acceptable No. of points = 3 Minimum search radius = 0.04

The dust data are then analyzed in Section 3.1.1. According to the results of these analyses, the LIOR technique can be a good candidate for dust filtering. Therefore, we developed the LIOR filter for dust which is explained in detailed in Section 3.1.2. Nonetheless, this filter has some limitations. As a result, Section 3.1.3 proposes a new filtering technique to address LIOR's shortcomings.

3.1.1 Data analysis method

The gathered data was analyzed to determine the properties of the measured point clouds. Dust points have an intensity range of 0 to 10 that is significantly lower than that of other objects. For instance, the point clouds are plotted according to their intensity values using the turbo colormap in Fig. 3.2(b), where the dust point's color is close to black, which corresponds to an intensity value of 0. There are also a few non-dust points, including some low-intensity ground points (dark blue). In Fig. 3.2(b), dust noise (disturbance caused by dust) accounts for approximately 4.55 percent of the total number of points that should be eliminated.

According to above observation, intensity is a viable criterion for classifying or filtering out dust point clouds. In the next step, the LIOR filter, which requires

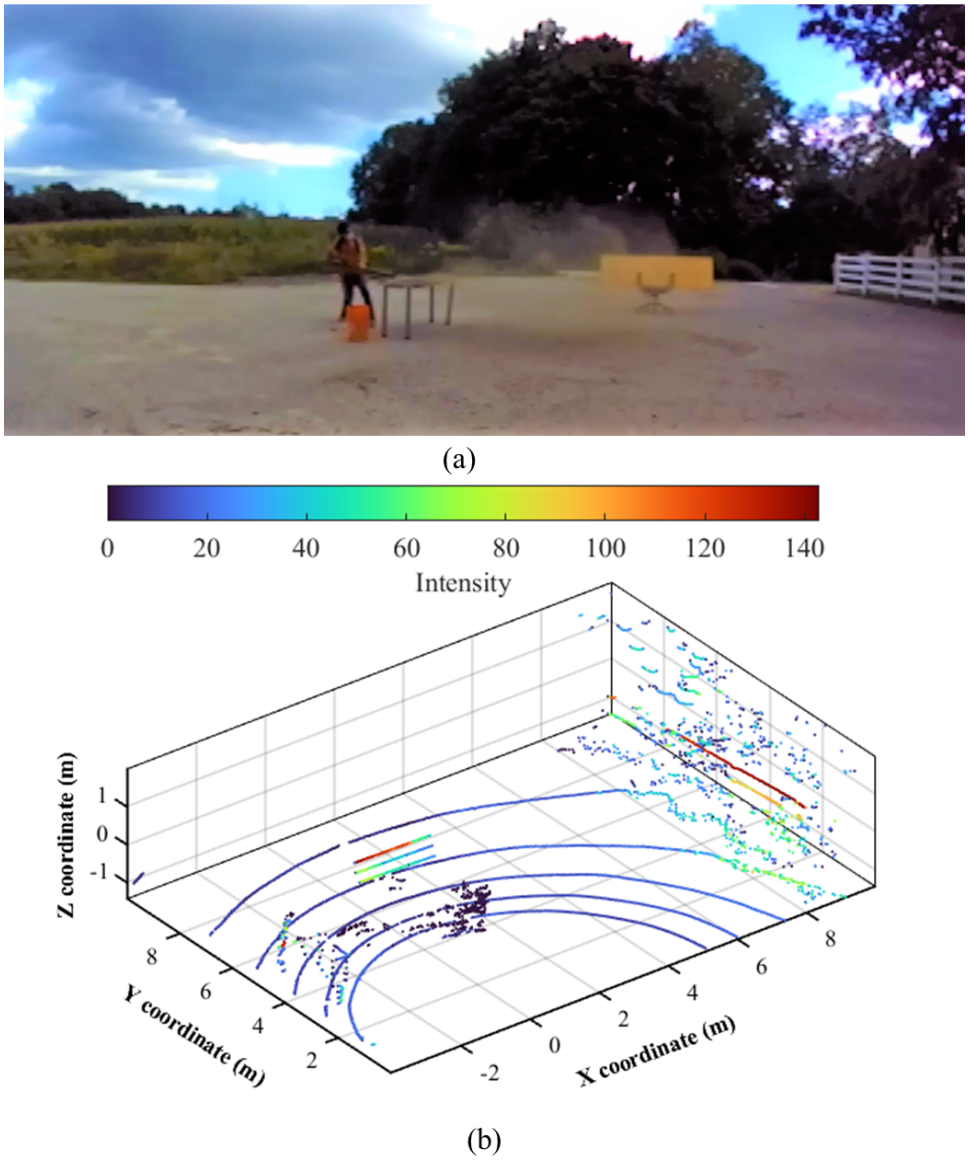
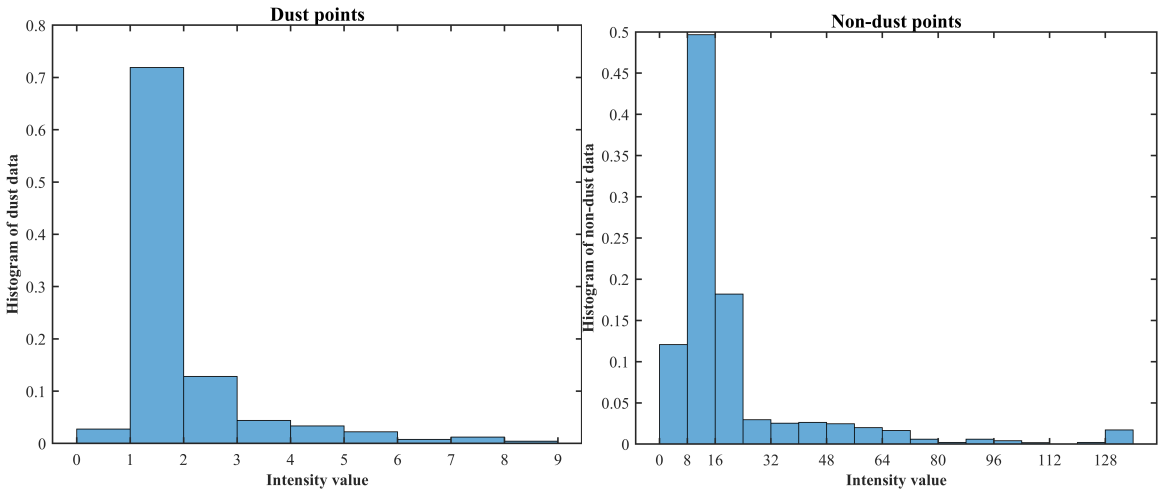


Figure 3.2: A scene of experimental data collection (a), and corresponding point cloud (b).

intensity data, is applied to evaluate its capability and efficacy at removing dust.



(a) (b)

Figure 3.3: Histogram of a VLP-16 LiDAR point clouds when exposed to dust: Histogram of dust points as a percentage of total dust points (a) and histogram of non-dust points as a percentage of total non-dust points (b).

3.1.2 Optimizing LIOR for de-dusting

The LIOR filter has three parameters, as described in Section 2.2.4: intensity threshold, search radius, and minimum acceptable number of points in the vicinity of a query point. Achieving a high-performance dust filter requires identifying the optimal intensity threshold value. Therefore, a data analysis was performed to determine the appropriate threshold intensity.

The histograms in Fig. 3.3 depicts the distribution of intensity values for dust and non-dust particles in Fig. 3.2. The intensity value in VLP-16 varies as an integer between 0 and 255. Specifically, the x-axis represents an integer intensity interval, whereas the y-axis represents a percentage of the intensity data falling within each interval. In Fig. 3.3a, for instance, the x value of the second bin is within the interval of [1,2), and its y value is approximately 71%. This indicates that 71 percent of dust points in Fig. 3.3a have an intensity equal to 1. In contrast, the vast majority of non-dust points, nearly 88 percent, have an intensity greater than 8 (see Fig. 3.3b).

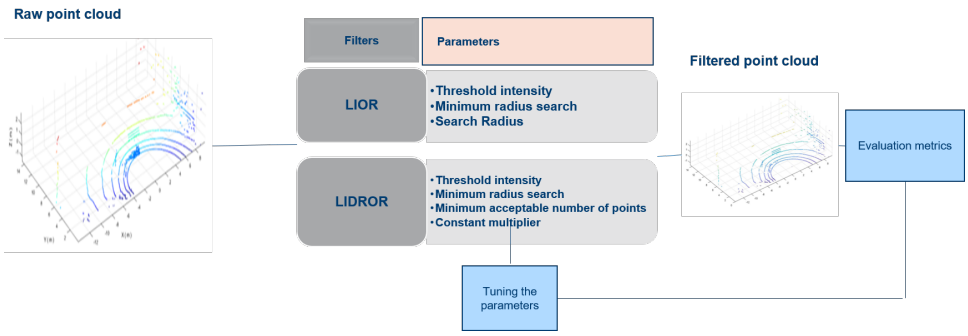


Figure 3.4: An illustration of the tuning process for LIOR and LIDROR.

A high threshold increases the possibility that low-intensity non-dust points will be removed. Consequently, there is a trade-off between dust removal and environmental information preservation, and both needed to be balanced. In the study, 7 was selected as the threshold intensity after considering both perspectives. As depicted in Fig. 3.4, the optimal values of the two remaining LIOR parameters, search radius, and minimum acceptable number of points, in Table 3.2 were determined by trial and error using the data sets described in Section 3.3.1.

Table 3.2: LIOR final parameters

LIOR parameters	value
Threshold intensity	7
Search radius (m)	0.044
Minimum acceptable number of points	6

3.1.3 Low-intensity dynamic radius outlier removal (LIDROR)

To make the LIOR filter more resistant to distance variation, we created a new filter called LIDROR. Specifically, the ROR filter was replaced by the DROR filter in the second stage of the LIOR filter in order to solve the ROR filter's problem by employing a dynamic search radius (from lines 5-9 of Fig. 3.5). Tuning parameters for de-dusting in this filter are the constant multiplier and the minimum acceptable

number of points within the search radius. They were tuned based on observations of how these parameters affect the filtering performance and robustness in various dust scenarios.

In addition to maximizing dust removal, the LIDROR filter has the advantage of allowing the threshold intensity to be set higher without sacrificing important non-dust information. Finding suggest that the threshold intensity for this filter should be 8, which is higher than the LIOR filter's threshold intensity of 7. Table 3.3 summarizes the finalized parameter values, including the threshold intensity.

Table 3.3: LIDROR final parameters

LIDROR parameters	value
Threshold intensity	8
Minimum radius search (m)	0.044
Minimum acceptable number of points	5
Constant multiplier	0.011

Algorithm 3 LIDROR filter

FOR (Each point in the point cloud)
  IF (point intensity $>$ threshold intensity)
    Inliers $\leftarrow$ point
  ELSE
    IF (search radius $<$ minimus search radius)
      search radius = minimum search radius
    ELSE
      Search radius $\leftarrow \phi \times \alpha \times \sqrt{x_p^2 + y_p^2}$
    ENDIF
    $n \leftarrow$ Find number of points inside SR
    IF ($n <$ threshold point)
      Outliers $\leftarrow$ point
    ELSE
      Inliers $\leftarrow$ point
    ENDIF
  ENDIF
ENDFOR

Figure 3.5: Pseudocode of the LIDROR filter.

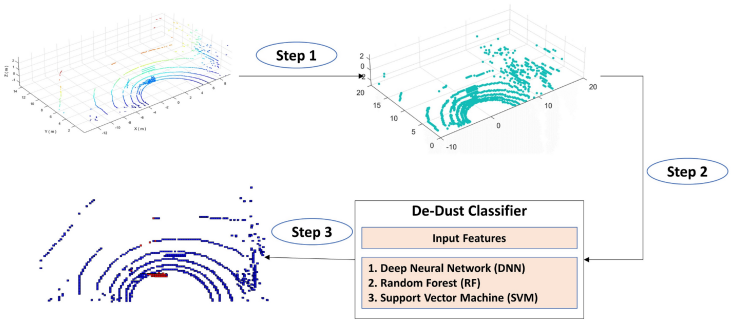


Figure 3.6: Illustration of the operation of AI classification methods.

3.2 Development of AI-based dust-filtering algorithm

This section suggests AI techniques for classifying LIDAR data from airborne particles. The classification techniques consist of a class for dust and one for non-dust. If every LIDAR point within a voxel is generated by dust particles, the voxel is labeled as dust. Figure 3.6 depicts the three-step classification method. The discretization of 3D LIDAR point clouds is the initial step. Second, the input features of each voxel are computed and passed on to the subsequent layer. Then, a machine learning classifier assigns a category to each voxel as the third step. Each of these step is explained in more detail in sections 3.2.1, 3.2.2, and 3.2.3, respectively.

3.2.1 Converting LiDAR point cloud into voxels

A 3D voxel map [24] is one of the most common techniques utilized in robotics for obstacle detection or classification from point clouds. In addition, the discretization of a point cloud is a common method for storing and processing spatial sensor data. In the study, LiDAR scans were discretized by dividing the three-dimensional space into equal-sized, non-overlapping voxels $[v_x, v_y, v_x]$. A voxel map with the dimensions

$[m_x, m_y, m_x]$ is centered on the LiDAR coordinates' origin. The experimentally determined dimensions of the voxel map are $m_x = [-10m, 10m]$, $m_y = [0m, 20m]$, and $m_z = [-3m, 3m]$. The voxel size was set to 0.2 m (i.e., $v_x = v_y = v_z = 0.2m$) so that each voxel contains enough LiDAR points to carry geometrical information while preserving sufficient scene details. Depending on their positions on the map, LiDAR points were accumulated in the corresponding voxels. Each LiDAR point is positioned relative to the map's origin. The intensity, a_i, is a number between 0 and 256 that represents the amount of light reflected by the LiDAR's surface hit. This value depends on several variables, including the distance to the object, the angle of incidence of the light ray on the surface, and the surface's material.

3.2.2 Feature selection method

To select the most effective features for dust classification, we narrowed the list of features to those that had previously been utilized in the literature and proven to be effective. These features include the mean intensity of the points within the voxel, the standard deviation of the points within the voxel, and a few other features that can be calculated using the eigenvalues obtained by performing PCA on the points within the voxel as discussed in Chapter 2.3.1. Then, to acquire a better understanding of these, each feature is plotted against other features, as shown in Fig. 3.7, where dust points and non-dust points are colored differently.

We selected input features for the classifier based on the analysis of patterns derived from Fig. 3.7. For the DNN network, the mean intensity of the points inside a voxel, the standard deviation of the points inside a voxel, the third eigenvalue, the planarity, and the curvature are selected as input features, while the mean intensity, the standard deviation, the slope and, the roughness are selected for RF and SVM classifier.

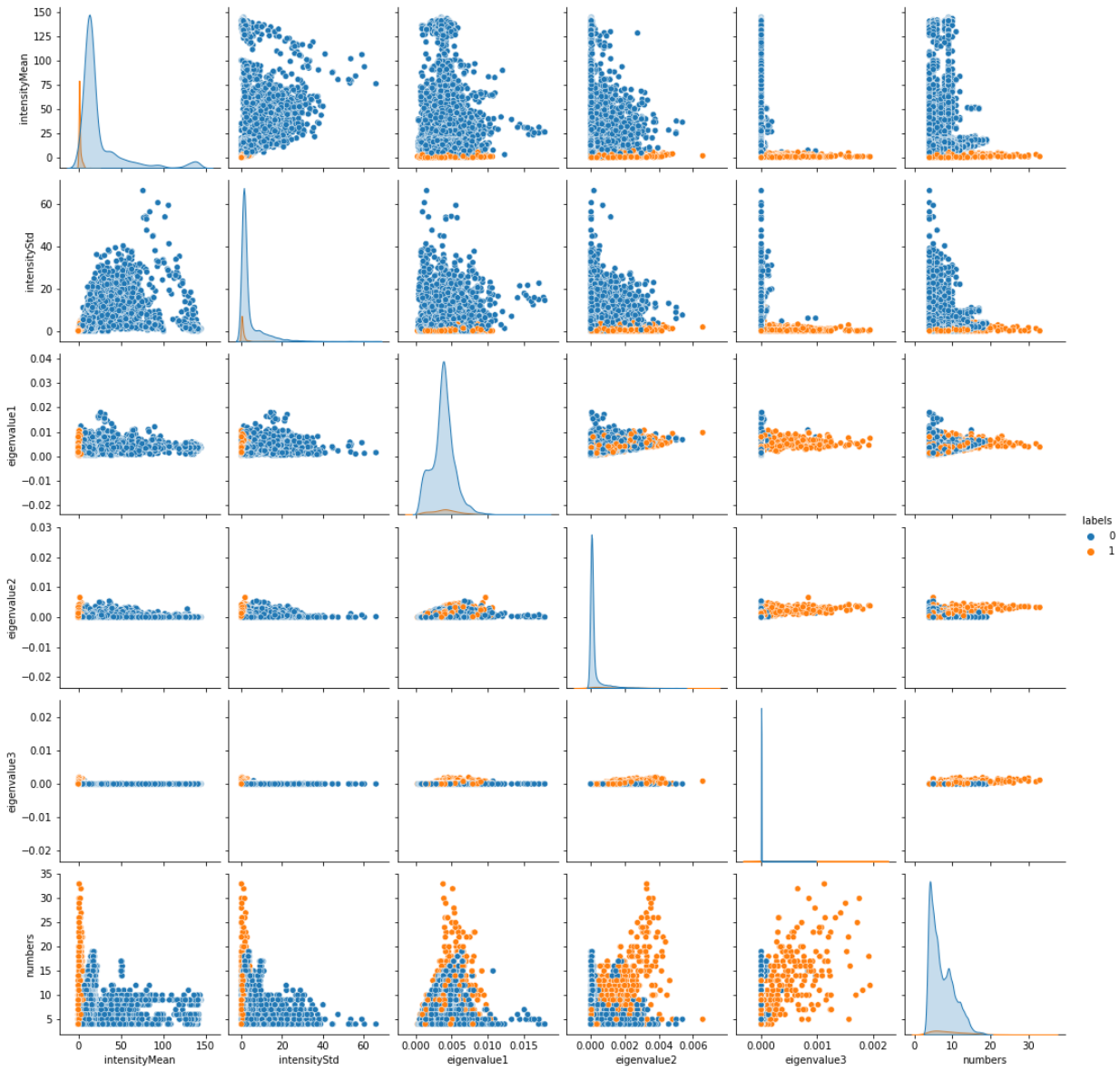


Figure 3.7: Candidate features are plotted against each other.

3.2.3 Machine learning technique used for de-dusting

As AI techniques, including Support Vector Machine (SVM), Random Forest (RF), and Deep Neural Network (DNN) were applied in this study. As mentioned in Sections 1.4.1, 1.4.2, and 1.4.4, these classifiers have some parameters that need to be tuned.

Different libraries were used to train these machine learning techniques. TensorFlow [57] and Keras [58] were used to tune the DNN hyperparameters, while Scikit-learn [59] was used to train the SVM and RF. For tuning the hyperparameters, we used the hyperband [60] method to find the best values for DNN. For SVM and RF, their parameters were empirically determined to obtain the best performance using the grid search function over a range of values and the 5-fold cross-validation step that is offered by the Scikit-learn library in Python. The hyperparameters that are used for this work are summarized in Table 3.4.

Table 3.4: Machine learning classifier final parameters.

Classifier	Parameters
SVM	Kernel: Gaussian RBF $\gamma = 10$ C = 400
RF	No. estimators = 10 Maximum depth = 15 Maximum features = 3 split criterion: Gini
DNN	No. hidden layers = 3 No. nodes in each layer = 128 Learning rate = 0.001 Optimization technique: NADAM

A Radial Basis Function (RBF) kernel with a penalty parameter of 400 and a kernel coefficient of 10 was utilized in the SVM classifier. For the Random Forest classifier, ten decision trees were utilized along with maximum depth and maximum features to prevent data overfitting. In addition, the Gini index was chosen as a splitting criterion. Three hidden layers containing 128 nodes were selected for DNN. The learning rate utilized by the NADAM optimization strategy for DNN is 0.001.

To train the mentioned classifiers, labeled LiDAR points that were acquired from

the first experiment in Section 3.3.1 were split into two groups with 80% training and 20% validation. The statistics related to first experiment data used for training is presented in Table. 3.5. We computed voxels from each cloud of LiDAR points, with a voxel size of 0.2 m.

Table 3.5: Statistics related to training data used for training the AI techniques

	No. voxels	
	Non-dust	Dust
training set	202,690	8,594
test set	50,672	2,148

3.3 Data preparation

A new data set with dust particles should be created to study the behavior of dust and evaluate the designed filter under the various scenarios. We created two distinct datasets to evaluate the developed algorithms. The first dataset contains a static scene and the LiDAR sensor is stationary, whereas the second dataset contains more complex experimental conditions such as a dynamic scene and robot movement.

3.3.1 First dataset

Data collection process

The experimental variables used in this dataset are based on the lesson learned from the paper [7]. Based on this paper, several parameters, including the distance between a LiDAR and dust clouds, the distance between a LiDAR and a target, the length of the dust cloud, the dust density, the dust particle size, and the reflectivity and the surface area of a target, influence LiDAR measurements exposed to dust. In

this dataset, the first two parameters (see Figure 3.8) were selected as design variables to generate a variety of experimental conditions. This is because the length, density, and size of the dust cloud are difficult to control. The reflectivity and the surface area were also omitted for the sake of simplicity, as computing these quantities for each point in the point cloud could complicate the problem. In this study, the distance between a LiDAR and the location of dust blowing was measured with a measuring tape, and the target was positioned at a predetermined location outlined in Table 3.6. As shown in this table, we designed four distinct experimental conditions by varying these two variables. Under these conditions, data were collected using a VLP-16 [30] LiDAR sensor and a leaf blower to generate dust particles on a clear day. Figure 3.10a depicts an experimental scene with a person, trees, and other background objects, as well as dust scattered by a blower. The LiDAR sensor used in this work is VLP-16 which has the following characteristics mentioned in Fig. 3.9.

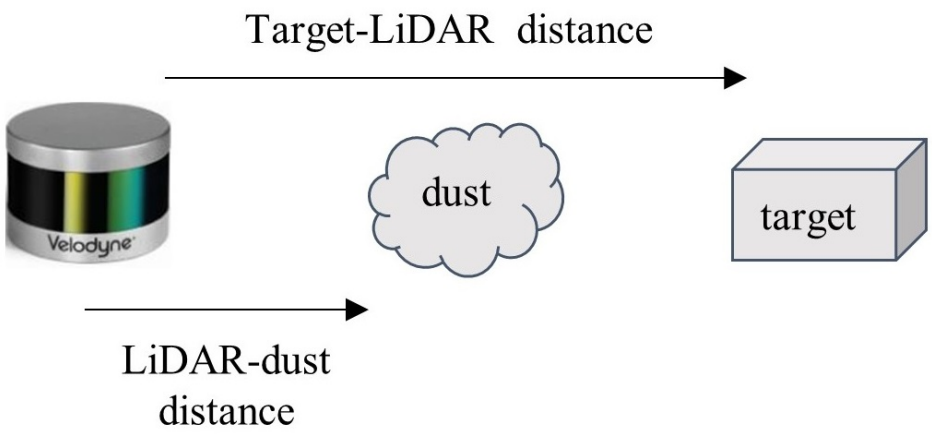


Figure 3.8: A diagram of the experimental variables. The design variables in this dataset are LiDAR-dust distance and LiDAR-target distance.

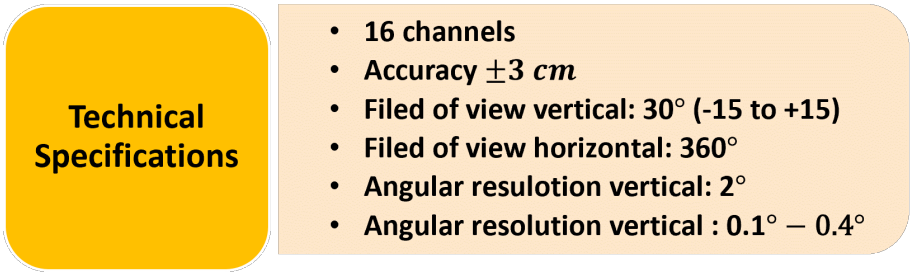
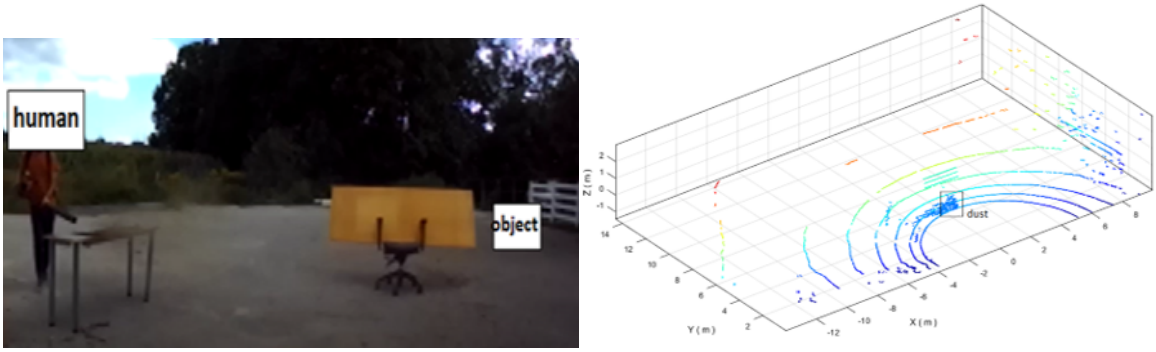


Figure 3.9: Technical specifications of VLP-16.

Table 3.6: Experimental conditions

No. Experiment	LiDAR-dust cloud distance	LiDAR-target distance
1	4	5
2	5	10
3	8	10
4	10	15



(a) (b)

Figure 3.10: Experimental scene of the first data set (a) and corresponding LiDAR point cloud (b).

3.3.2 Second dataset

The second dataset includes more sophisticated situations. The goal of this dataset is to assess the robustness of the designed AI-technique-based filters against more complex scenarios. These scenarios include the following components:

1. Sensor location changes as the mobile robot is moving.

2. A person is walking around the scene, acting as a moving object.

3. The density of dust varies depending on whether one or two blowers are used. One blower represents low-density dust, whereas two blowers represent high-density dust.

To meet the first component in the second dataset, a mobile platform is required. As a result, we designed our own platform to satisfy the following criteria:

1. The LiDAR sensor should be mounted in such a way that the LiDAR emissions do not collide with the platform.

2. This platform should have enough space to accommodate the laptop, as all computation is currently done on the laptop.

3. A depth camera should be installed on this platform to capture the scene situations that are required for data labeling.

4. This platform must be able to support the weight of the sensors, battery, laptop, and frame that will be mounted on it.

Mobile platform design

After conducting extensive research on different platforms on the market, the 1/10 Night Crawler RC vehicle depicted in Fig. 3.11 was purchased to meet the aforementioned requirements. However, how to effectively collocate the LiDAR sensor, depth camera, and other essential components on an RC car is another designing element.

To tackle this problem, we created the structure depicted in Fig. 3.12. This structure has three levels. It is connected to the RC vehicle on the first floor, and the laptop is also placed on it. The LiDAR sensor is installed on the top floor. The depth

Figure 3.11: A picture from the RC car used in this work.

camera is positioned on the second level, along with other essential equipment such as the power source for the LiDAR sensor. The design is then printed on a 3D printer and assembled on an RC car. The final result is illustrated in Fig. 3.13.

Data collection process

The design variable for second dataset are presented in Table 3.7. The dust density in this data set varies depending on whether one or two blowers are utilized. In each experiment, a human walks as a dynamic object around the scene. Other variables in this experiment included the initial distance from the LiDAR sensor to the dust cloud and target.

3.3.3 Ground truth labeling

Both training AI algorithms and evaluating proposed filters require factual data. In particular, ground truth data was labeled using the MATLAB LiDAR Labeler app based on the data collector's prior knowledge and experience with the experiment.

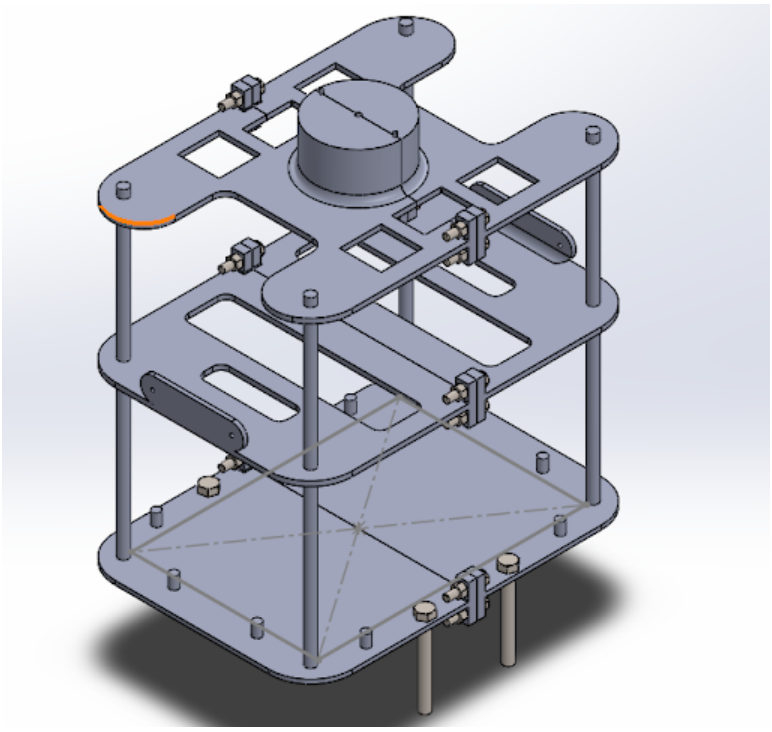


Figure 3.12: Designed structure in Solidworks. This structure is mounted on the RC car.

Table 3.7: Experimental conditions

No. Experiment	Initial LiDAR-dust distance	Initial LiDAR-target distance	No. of blowers
1	4	5	1 or 2
2	6	5	1 or 2
3	7	10	1 or 2
4	11	10	1 or 2

Fig. 3.15 depicts the MATLAB LiDAR Labeler's environment for the labeling process. This app's user interface for labeling allows users to easily create cuboids containing dust points (particles). This application also permits users to calculate the cuboid's center and dimensions (length, width, and height). Using this data and a MATLAB script, we could determine which LiDAR points fall within the dust cuboids and label them as either dust or non-dust.

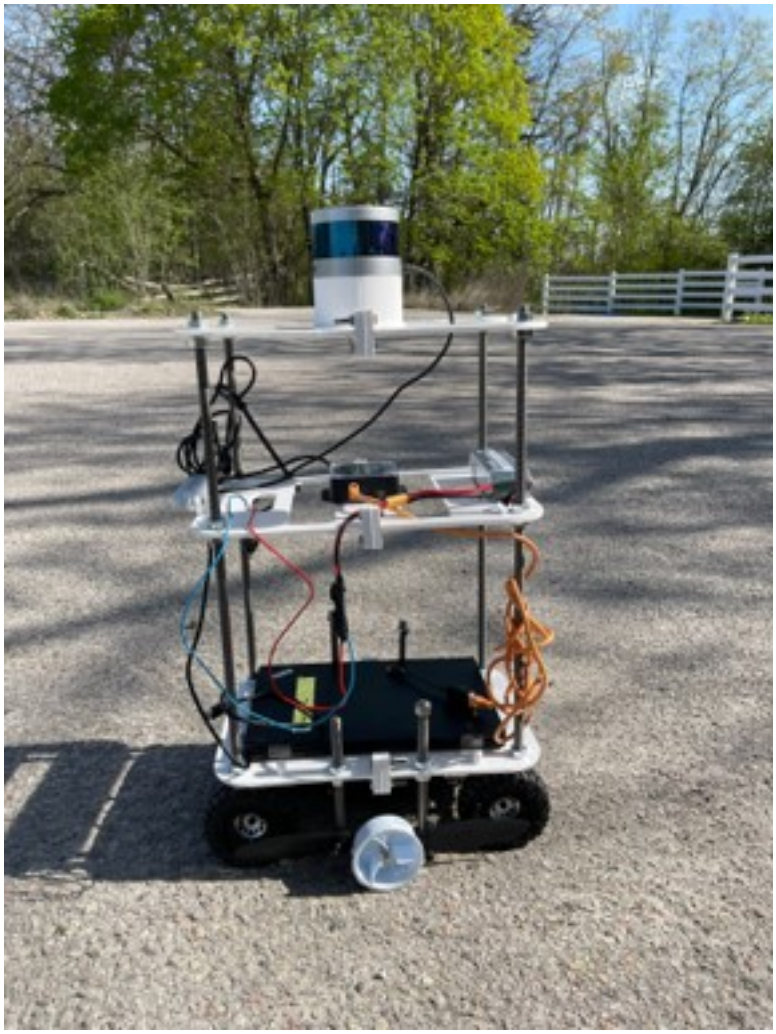


Figure 3.13: Designed mobile platform used for gathering the data in the second dataset.

3.4 Integrating into ROS environment

The developed filtering algorithms need to be deployed in real-time for on-site testing. To do this, all programs and training models were integrated into the ROS environment. Fig. 3.16 shows the architecture of the developed codes in the ROS environments. This architecture consists of three nodes. The first node, Velodyne_points, is responsible for collecting data from the LiDAR and depth camera. The objective of the second node is to convert the point cloud into voxels and then compute the re-

Figure 3.14: A picture of the scene when data was gathered for the second dataset.

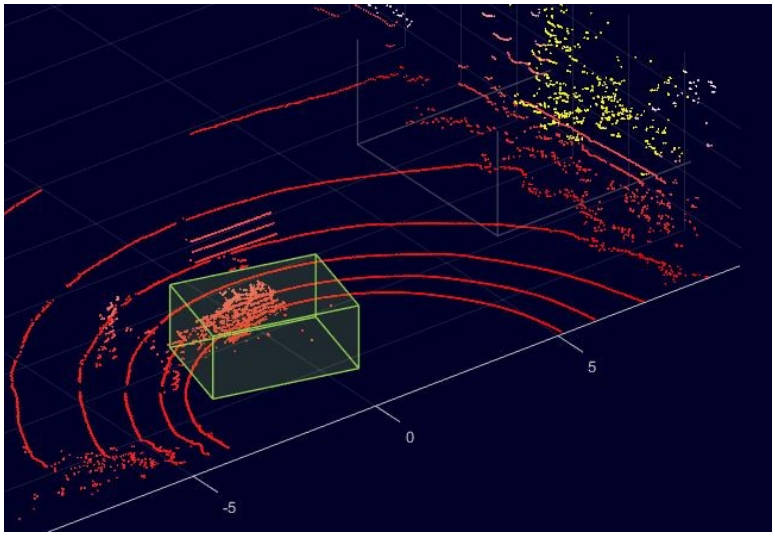


Figure 3.15: An example of a LiDAR labeler app in the MATLAB environment.

quired features for classification. The name of this node is feature_node classVersion. The final node, named as rostensorflow, aims to predict and visualize the results.

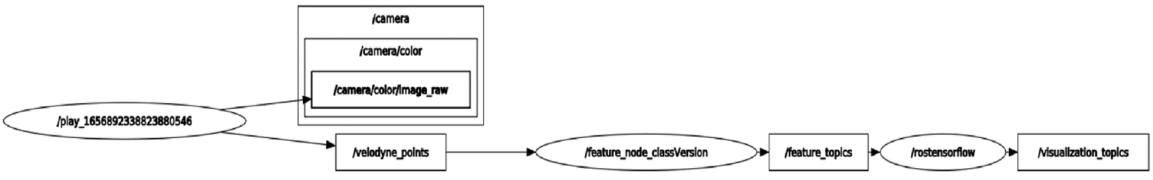


Figure 3.16: Illustration of ROS nodes used in this work and their dependencies.

3.4.1 Voxel Map

The first step in implementing these algorithms in practice is to get the data from the LiDAR sensor and convert these data into voxels. For the first part, a Velodyne Library [61] in ROS was used. This library includes Velodyne driver, Velodyne laser scan, Velodyne msgs, Velodyne PCL, and Velodyne point cloud as five separate packages. Each library has its unique functionality. However, two of them are useful for this work: Velodyne driver and Velodyne pointcloud.

Velodyne driver is responsible for taking the data from the Velodyne and combining it into one message per revolution. These data are still raw and cannot be used by the application. These data must be processed by another package called Velodyne pointcloud in order to obtain the Cartesian information such as the coordinates of points in Cartesian space. This driver converts raw Velodyne data produced by the Velodyne driver node into a PointCloud2 message.

C++ programming was selected for creating voxels with the size of 0.2 m and computing the features based on voxels because it is the most efficient programming language available. A crop filter was applied to the data to focus only on the region of interest. In order to use the point cloud data as input features for the proposed artificial intelligence techniques, it should first be converted to a voxelized format. To create the voxels, octree data structure from Point Cloud Library (PCL) [62] was used. Finally, the eigenvalues, eigenvectors, mean intensity, and standard deviation intensity of the points inside voxels were computed using the Singular Value Decomposition function in the Eigen library [63], a C++ library that provides some linear algebra operations.

3.4.2 Visualization

In this work, it is necessary to visualize various data coming from the sensors. The first is the raw data collected by LiDAR sensors. The second is a video from a depth camera recording the scene. Finally, we should display the voxelized cube prior to and after filtering. RVIZ was used for all of these visualizations because it is quick and efficient when working with ROS.

There are packages available in RVIZ for visualizing LiDAR and cameras. To visualize the voxels, however, programming is required. RVIZ's Marker array cube list, which was used in this study, is a custom data type optimized for the simultaneous visualization of multiple cubes. Fig. 3.17 is an illustration of this type of visualization. The upper left window displays classification results, with green cubes representing non-dust points and yellow cubes representing dust points. The window on the upper right is the result of removing the dust voxels. The lower left window is the raw

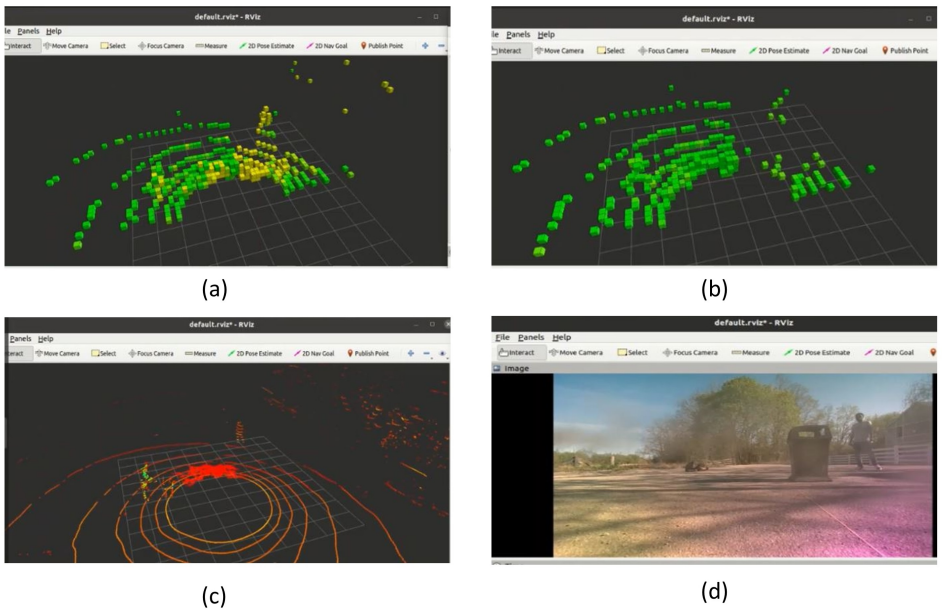


Figure 3.17: Voxel visualization of data in RVIZ: classified voxels before filtering (a), and classified voxels after filtering (b), and raw LiDAR point cloud data (c), and image of the scence captured by a depth camera (d).

LiDAR point cloud data, while the lower right window is the captured depth camera image.

Chapter 4

Discussion and Results

The objective of this chapter is to present the outcomes of applying the suggested filters. First, the metrics that were used for the evaluation of this work is explained. Then, the results of non-AI techniques are presented first. After that, the outcomes of AI-based techniques are presented.

4.1 Evaluation metrics

The data collected in the first and second experiments were labeled to evaluate the proposed filters. Labeling was accomplished using the LiDAR labeler application in MATLAB [64] as illustrated in Fig. 4.1, which allows us to draw a cube around the dust cloud and label it as dust. The points inside the yellow cube are labeled as dust in the figure. Dust and non-dust point clouds are consequently labeled 1 and 0, respectively. Accuracy, precision, recall, and F1-score, as defined in Equations

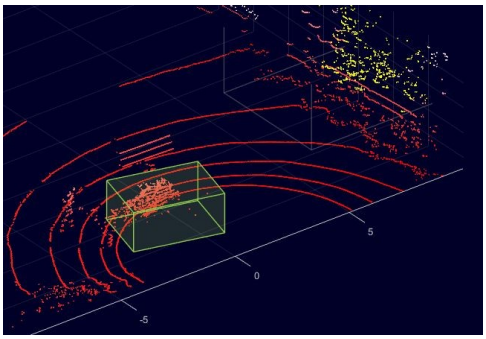


Figure 4.1: Labeling data in MATLAB LiDAR labeler app.

(4.1)-(4.4), are the metrics used to evaluate the performance of the developed filters.

$$Accuracy = \frac{TP + TN}{N} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}} \tag{4.4}$$

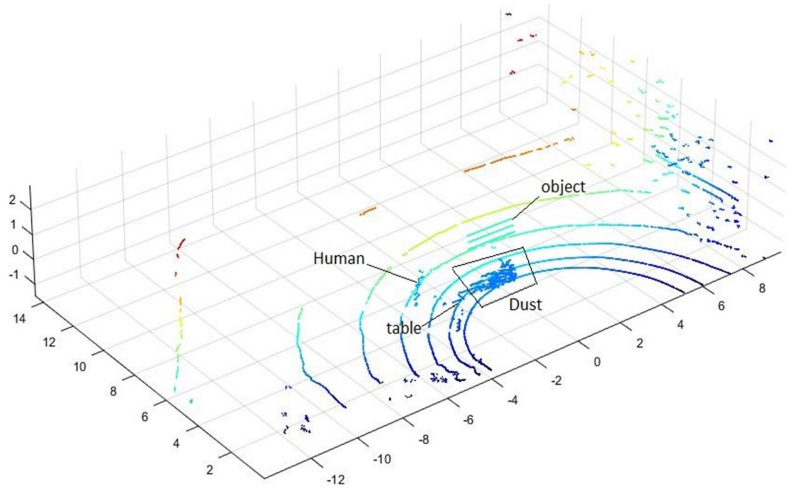
where TP represents the number of dust points that are removed correctly, TN represents the number of non-dust points that are saved correctly, FP represents the number of non-dust points that are removed as dust falsely, FN represents the number of dust points that are preserved as non-dust falsely, and N represents the total number of points inside the point cloud. A high precision score indicates a low FP, indicating that the filter effectively eliminates dust noise. A high recall score, on the other hand, indicates a low FN, indicating that the filter can effectively preserve environmental information.

4.2 Non-AI techniques

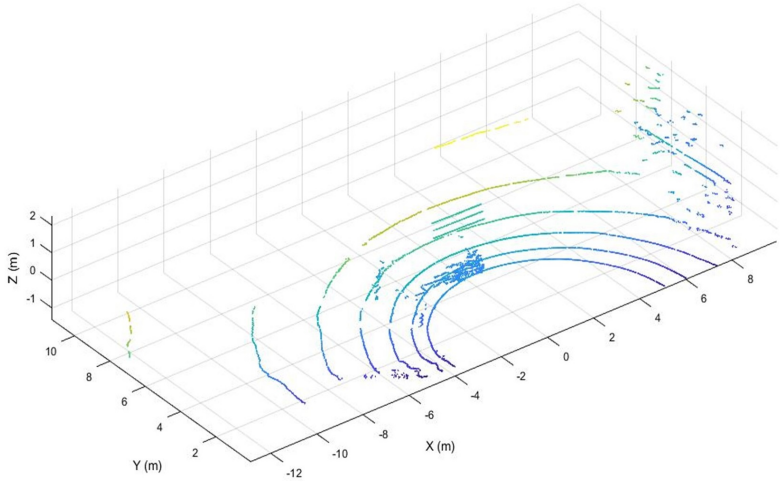
The results of dust removal with the designed filters SOR, ROR, and DROR are depicted in Fig. 4.2-Fig. 4.4 for one of the scenario equivalent to experiment No. 1 in Table 3.6. In this scenario, dust clouds are located approximately 4 m from a LiDAR sensor. For LIOR and LIDROR, another scenario was added for testing their robustness under different conditions. The additional scenario is from Table 3.6 in which dust clouds are located within 8 m (experiment No. 3 in Table 3.6). The results for LIOR and LIDROR are presented in Fig. 4.5 and Fig. 4.6. In all the figures, the left figures are before filtering, and the right figures are after filtering.

Furthermore, the evaluation metrics described in Section 4.1 were applied to the first scenario corresponding to experiment No.3 in Table 3.6. The results are shown in Table 4.1. The SOR filter has the worst overall performance for removing dust noise, which accounts for 4 percent of the total point cloud, according to the evaluation results with the four metrics, as shown in Table 4.1. The SOR filter, on the other hand, has a higher accuracy value than the ROR filter. This filter is ideal for removing noises that are isolated from others because the SOR only considers the k-nearest points when removing outliers (i.e., removing sparse outliers). This filter is ineffective at removing dust, however, because the dust point cloud consists of a very small number of isolated points.

On the other hand, the performance of the ROR filter for removing dust is dependent on the selected search radius, as a small search radius results in the loss of significant environmental information. The ROR outperforms the SOR because it considers the neighborhood density. By addressing the sparsity issue in the LiDAR point cloud, the DROR filter produces better results than the ROR and SOR. Due to the same limitation as the ROR, selecting a smaller search radius than the current



(a)

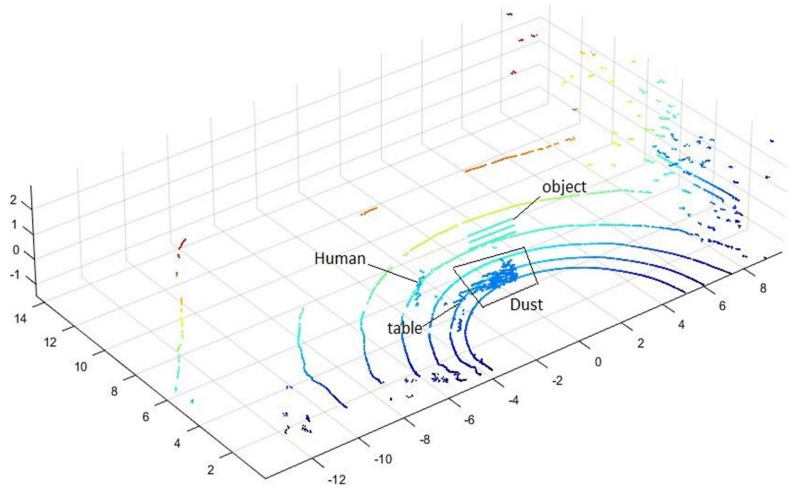


(b)

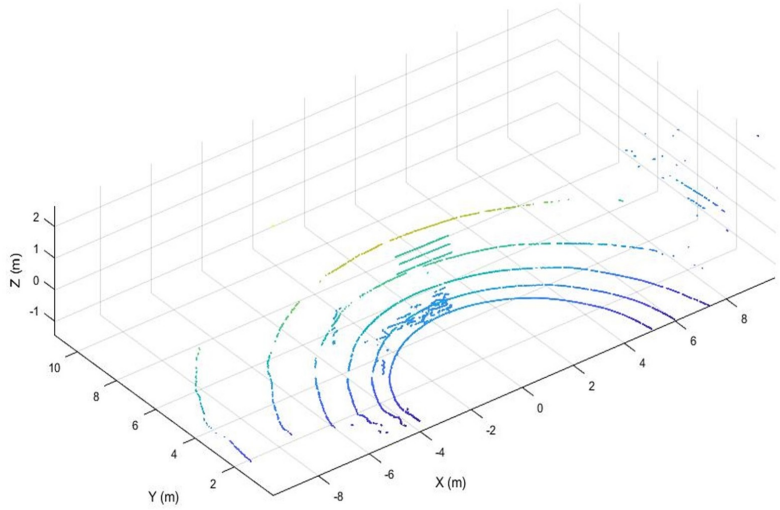
Figure 4.2: Experimental results following the application of the developed SOR de-dusting filter: Point cloud map prior to filtering in case of experiment No.1 in Table 3.6 (a), and point cloud map after SOR filtering in case of experiment No.1 in Table 3.6 (b).

one cannot improve the performance of de-dusting filter.

Regarding dust removal, the LIOR filter is comparable to the LIDROR filter. Due to the sparseness of a LiDAR point cloud at long range and considering the point that the LIOR eliminates nearly all non-dust points selected in the first step beyond a



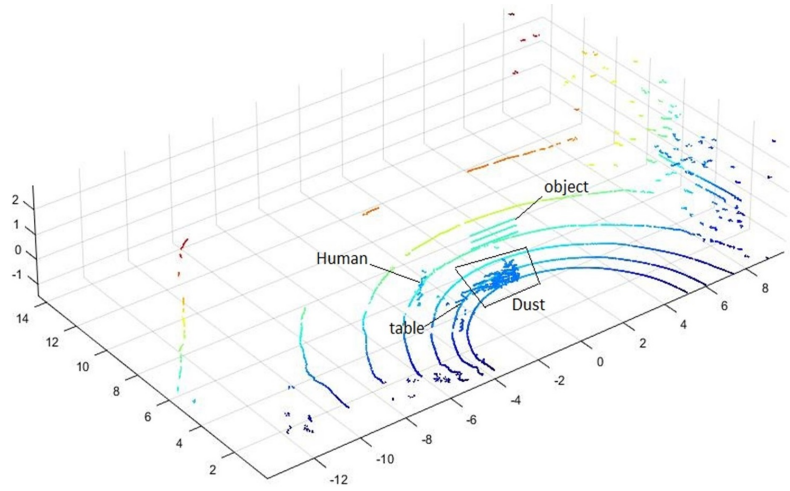
(a)



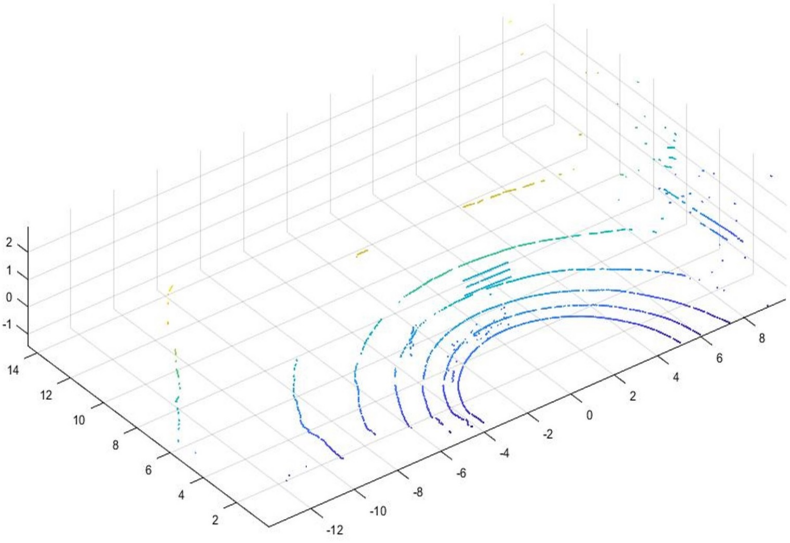
(b)

Figure 4.3: Experimental results following the application of the developed ROR dedusting filter: Point cloud map prior to filtering in case of experiment No.1 in Table 3.6 (a), and point cloud map after ROR filtering in case of experiment No.1 in Table 3.6 (b).

certain distance, it has a lower recall score than LIDROR. The LIDROR has the best performance across all metrics among the five filters, with an exceptional F1-score of 97.55 percent. In addition, it has the highest recall value (95.74 percent) and a preci-



(a)



(b)

Figure 4.4: Experimental results following the application of the developed DROR de-dusting filter: Point cloud map prior to filtering in case of experiment No.1 in Table 3.6 (a), and point cloud map after DROR filtering in case of experiment No.1 in Table 3.6 (b).

sion value near 100 percent, indicating that this filter is not only capable of preserving environmental data, but also of removing nearly all dust from the point cloud. Although LIDROR has a higher *F1*-score than LIOR, it is more expensive to compute.

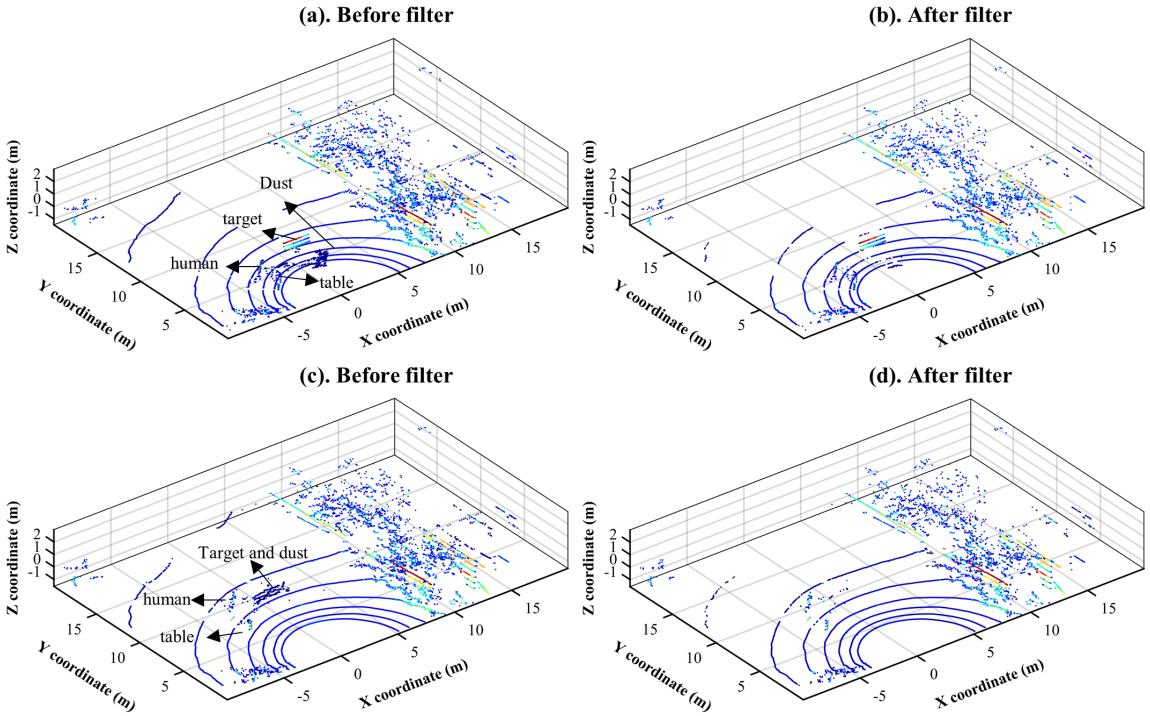


Figure 4.5: Results for LIOR de-dust filter: point cloud map before filtering in case of experiment No.1, first scenario (a), and point cloud map after filtering in case of experiment No.1, first scenario (b). Point cloud map before filtering in case of experiment No.3, second scenario (c), and point cloud map after filtering in case of experiment No.3, second scenario (d).

For the LIOR and LIDROR, the processing time for filtering is approximately 0.383 and 0.412 seconds, respectively.

4.3 AI techniques

The DNN technique was applied to one of the frames from the second dataset corresponding to No. 1 in Table 4.2, while the RF and SVM techniques were applied to the same first scenario as non-AI techniques. Fig. 4.7 and 4.8 show the results for SVM and RF, respectively.

For instance, based on Fig. 4.7 and Fig. 4.8, it can be seen that the SVM incorrectly classifies some dust points, whereas the RF correctly classifies them. Random forest

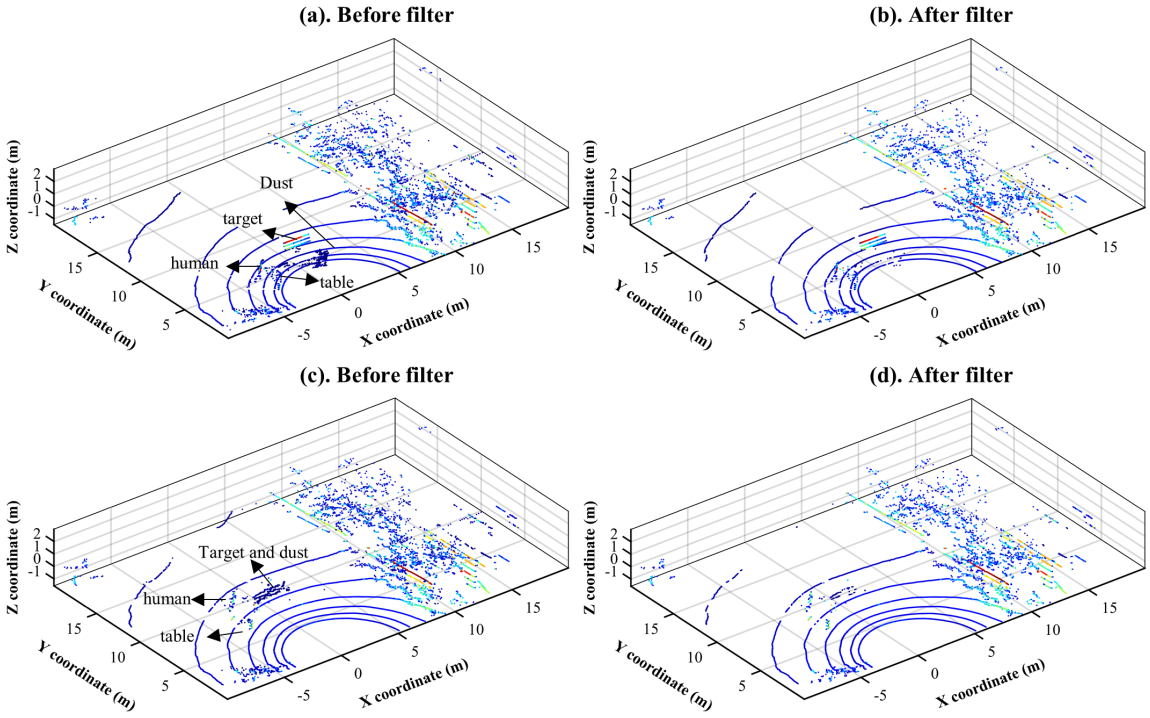


Figure 4.6: Results for LIDROR de-dust filter: point cloud map before filtering in case of experiment No.1, first scenario (a), and point cloud map after filtering in case of experiment No.1, first scenario (b). Point cloud map before filtering in case of experiment No.3, second scenario (c), and point cloud map after filtering in case of experiment No.3, second scenario (d).

Table 4.1: Evaluation results for non-AI technique

Filters	Evaluation metrics (%)			
	Accuracy (%)	Precision (%)	Recall (%)	*F1*-score (%)
SOR	86.3	0.21	0.33	0.26
ROR	73.11	10.77	54.25	17.97
DROR	91.63	36.78	75.49	49.46
LIOR	89	99.27	89.87	94.24
LIDROR	95.46	99.44	95.74	97.55

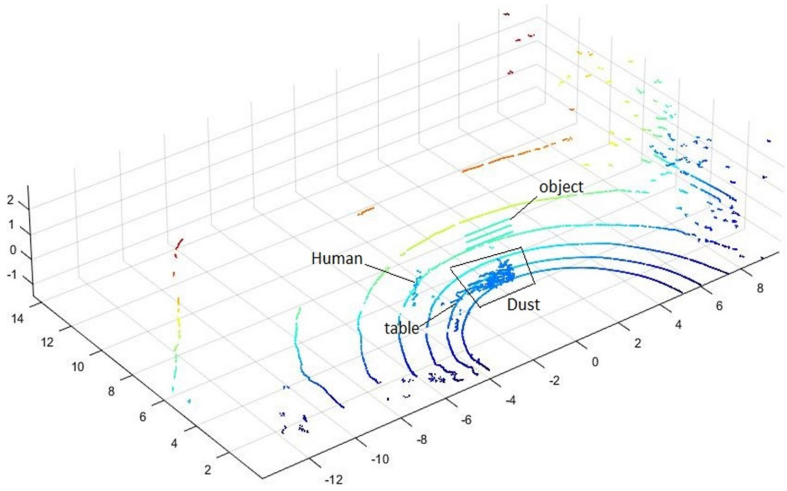
Table 4.2: Evaluation results for SVM and RF

Filters	Evaluation metrics (%)			
	Accuracy (%)	Precision (%)	Recall (%)	*F1*-score (%)
SVM	96.86	81.48	75.86	78.57
RF	98.43	92.5	86.2	89.2

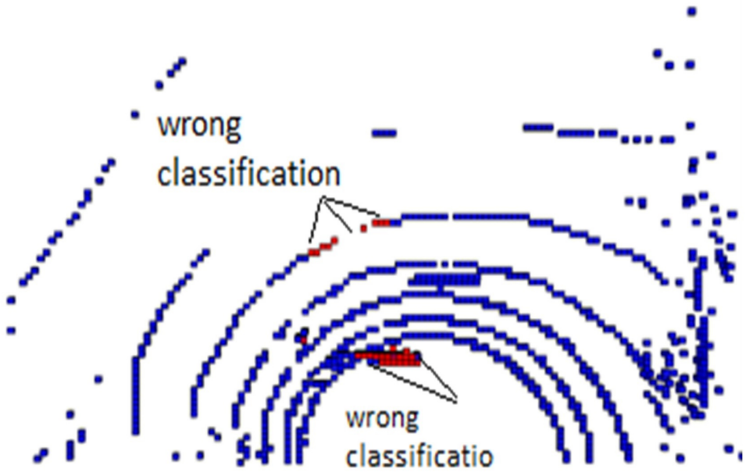
Table 4.3: Evaluation results for DNN

Filters	Evaluation metrics (%)			
	Accuracy (%)	Precision (%)	Recall (%)	*F1*-score (%)
1st dataset	97.2	87.95	79.08	83.28
2nd dataset	92.93	81.15	73.46	77.11

trains a group of decision tree classifiers, which is one of the reasons why it performs better than SVM. To make predictions, it collects the predictions of all tree nodes and then predicts the class with the most votes. It turns out that combining the predictions of multiple predictors can result in a more accurate forecast than using the best predictor alone. The DNN classifier shows the F1-score less than RF. It needs to mention that DNN are applied to two different datasets. The second dataset has a more sophisticated scenario, including a moving sensor and dynamic objects in the scene, making the prediction much more difficult. Consequently, the DNN performs better on the first dataset. However, the DNN's score for second dataset is 77.11, still



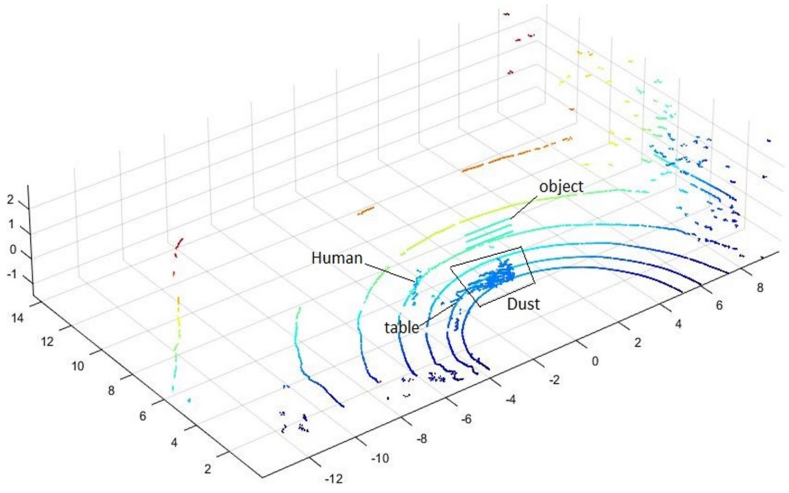
(a)



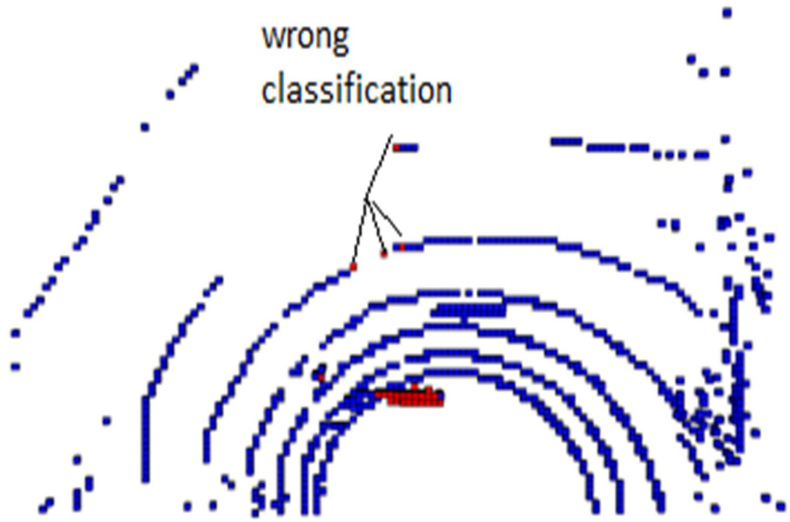
(b)

Figure 4.7: Experimental results following the application of the developed SVM classification: Point cloud map prior to filtering in case of experiment No.1 in Table 3.6 (a), and point cloud map after SVM classification in case of experiment No.1 in Table 3.6 (b).

allowing it to be applied to more complex situations. Fig. 4.9 and Fig. 4.10 illustrate the DNN results applied to the first and second datasets, respectively.

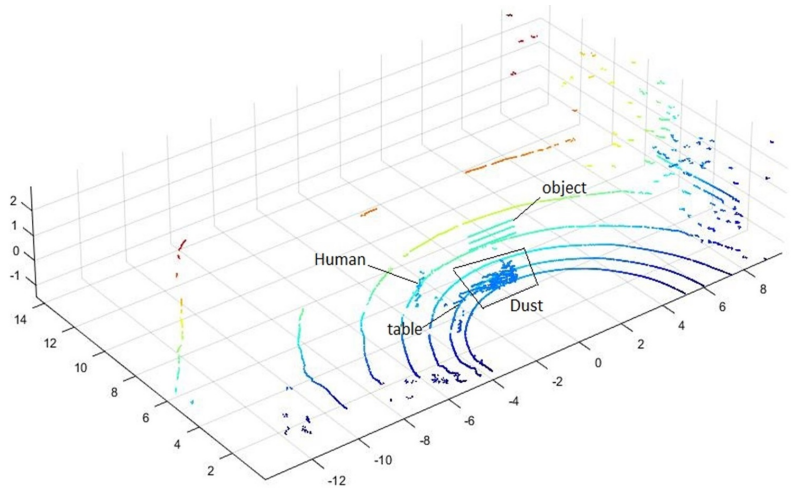


(a)

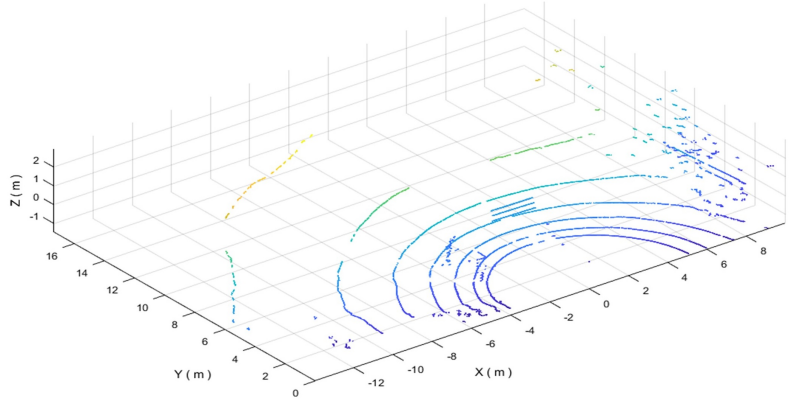


(b)

Figure 4.8: Experimental results following the application of the developed RF classification: Point cloud map prior to filtering in case of experiment No.1 in Table 3.6 (a), and point cloud map after RF classification in case of experiment No.1 in Table 3.6 (b).

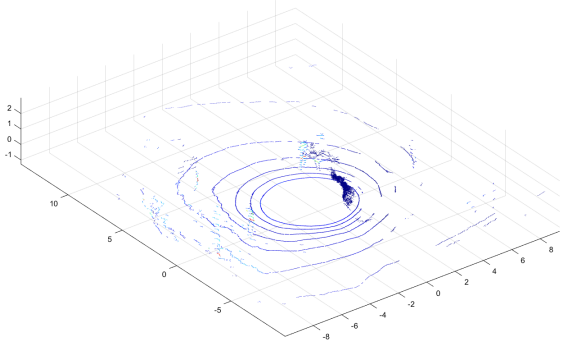


(a)

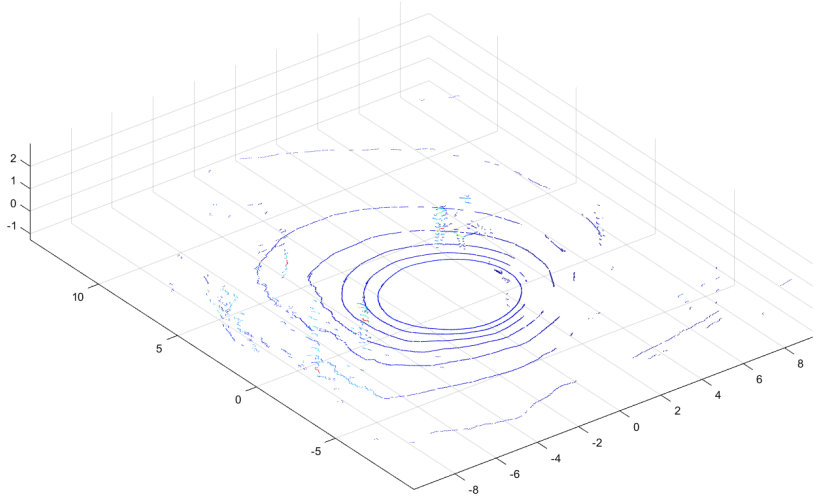


(b)

Figure 4.9: Experimental results following the application of the developed DNN classification: Point cloud map prior to filtering in case of experiment No.1 in Table 3.6 (a), and point cloud map after DNN classification in case of experiment No.1 in Table 3.6 (b).



(a)



(b)

Figure 4.10: Experimental results following the application of the developed DNN classification: Point cloud map prior to classification in case of experiment No.1 in Table 3.7 (a), point cloud map after DNN classification in case of experiment No.1 in Table 3.7 (b).

Chapter 5

Conclusions and future work

5.1 Conclusion

This thesis aims to design noise-filtering algorithms that can eliminate dust from LiDAR data for mobile industrial machines operating in dusty environments. In order to reach the objective, we developed de-dust filters with two different approaches including non-AI techniques and AI techniques. Non-AI techniques considered in this work include Statistical Outlier Removal Filter (SOR), Radius Outlier Removal Filter (ROR), Dynamic Radius Outlier Removal Filter (DROR), Low-Intensity Outlier Removal Filter (LIOR) and Low-Intensity Dynamic Outlier Removal Filter (LIDROR). The intensity-based filter, LIOR and LIDROR, were developed based on a comprehensive analysis of the properties of dust point clouds measured by a LiDAR sensor.

Among the AI techniques, we also selected some of the algorithms that have shown good performance for adverse weather conditions and then designed them for the case of dust. These algorithms include Support Vector Machine (SVM), Random Forest (RF) and Deep Neural Network (DNN).

To evaluate the developed de-dusting algorithms, two different datasets are col-

lected. These datasets were then labeled manually using prior knowledge about the data gathering. Then four different metrics were used with the manually labeled data sets to validate the performance of the developed algorithms.

For non-AI techniques, evaluation results show that the proposed LIOR and LIDROR filters outperform the conventional filters, including SOR, ROR, and DROR. Moreover, the LIDROR provides the most accurate and robust performance for dust removal with an $F1$-score of 97.55. For the AI technique, although the RF has the highest *F1*-score among them, the DNN was applied to a more sophisticated dataset which means that in overall, the DNN and RF have a good performance for dust filtering. Furthermore, all the developed filter in this work was implemented in ROS for real-time applications. It is expected from the results that the proposed filters can be used in applications such as mining and off-road machinery under harsh environmental conditions with dust.

5.2 Contribution

The contribution of this research can be summarized as follows:

- The suggested non-AI methods are, to the best of our knowledge, the first attempt to construct dust-filtering algorithms utilizing non-AI techniques that use the intrinsic properties (intensity value) of dust point-cloud data.

- The suggested non-AI solution can tackle the fundamental problems of AI dust-filtering methods, which need a large number of training data sets, resulting in high computing costs and training time.

- This paper discusses various design methodologies using SOR, ROR, DROR, LIOR, LIDROR, and AI techniques such RF, SVM, and DNN. As a result of the

comparison study, it provides practical advice on which approach is the most appropriate to remove dust from LiDAR sensor data.

5.3 Future work

1. One of the limitation of current non-AI filter is that they were tested on a limited dust conditions. So, future work for non-AI techniques can involve testing them in a greater variety of dynamic scenarios (e.g., varying dust conditions such as dust density, including moving objects to detect, etc.).

2. For non-AI technique, Convolutional Neural Network (CNN) can be considered as an alternative for designing a dust filter as a future work.

3. The next step in this project that could be beneficial is to determine whether combining LiDAR and camera data can effectively solve the de-dusting problem.

Bibliography

[1] Nathan Melenbrink, Justin Werfel, and Achim Menges. On-site autonomous construction robots: Towards unsupervised building. *Automation in construction*, 119:103312, 2020.

[2] Hadi Ardiny, Stefan Witwicki, and Francesco Mondada. Construction automation with autonomous mobile robots: A review. In *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, pages 418–424, 2015. doi: 10.1109/ICRoM.2015.7367821.

[3] Tyson Govan Phillips, Nicky Guenther, and Peter Ross McAree. When the dust settles: The four behaviors of lidar in the presence of fine airborne particulates. *Journal of field robotics*, 34(5):985–1009, 2017.

[4] Velodyne lidar. `https://velodynelidar.com/products/puck/`, . Accessed: 2022-07-14.

[5] Desheng Xie, Youchun Xu, and Rendong Wang. Obstacle detection and tracking method for autonomous vehicle based on three-dimensional lidar. *International Journal of Advanced Robotic Systems*, 16(2):1729881419831587, 2019.

[6] Hualei Zhang and Mohammad Asif Ikbal. Unmanned vehicle dynamic obstacle

detection, tracking and recognition method based on laser sensor. *International Journal of Intelligent Computing and Cybernetics*, 2021.

[7] Ruike Ren, Hao Fu, Hanzhang Xue, Xiaohui Li, Xiaochang Hu, and Meiping Wu. Lidar-based robust localization for field autonomous vehicles in off-road environments. *Journal of Field Robotics*, 38(8):1059–1077, 2021.

[8] Ilya Belkin, Alexander Abramenko, and Dmitry Yudin. Real-time lidar-based localization of mobile ground robot. *Procedia Computer Science*, 186:440–448, 2021.

[9] Jean-François Lalonde, Nicolas Vandapel, Daniel F Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of field robotics*, 23(10):839–861, 2006.

[10] Stefan Laible, Yasir Niaz Khan, Karsten Bohlmann, and Andreas Zell. 3d lidar- and camera-based terrain classification under different lighting conditions. In *Autonomous Mobile Systems 2012*, pages 21–29. Springer, 2012.

[11] Leo Stanislas, Julian Nubert, Daniel Dugas, Julia Nitsch, Niko Sünderhauf, Roland Siegwart, Cesar Cadena, and Thierry Peynot. Airborne particle classification in lidar point clouds using deep learning. In *Field and Service Robotics*, pages 395–410. Springer, 2021.

[12] Robin Heinzler, Florian Piewak, Philipp Schindler, and Wilhelm Stork. Cnn-based lidar point cloud de-noising in adverse weather. *IEEE Robotics and Automation Letters*, 5(2):2514–2521, 2020.

[13] Nicholas Charron, Stephen Phillips, and Steven L Waslander. De-noising of lidar point clouds corrupted by snowfall. In *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 254–261. IEEE, 2018.

[14] Abu Ubaidah Shamsudin, Kazunori Ohno, Thomas Westfechtel, Suzuki Takahiro, Yoshito Okada, and Satoshi Tadokoro. Fog removal using laser beam penetration, laser intensity, and geometrical features for 3d measurements in fog-filled room. *Advanced Robotics*, 30(11-12):729–743, 2016.

[15] Mario Bijelic, Tobias Gruber, Fahim Mannan, Florian Kraus, Werner Ritter, Klaus Dietmayer, and Felix Heide. Seeing through fog without seeing fog: Deep multimodal sensor fusion in unseen adverse weather. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11682–11692, 2020.

[16] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of field Robotics*, 25(8):425–466, 2008.

[17] Support vector machine (svm). `https://dinhanhthi.com/support-vector-machine/`, . Accessed: 2022-05-14.

[18] Svm using scikit-learn in python. `https://learnopencv.com/svm-using-scikit-learn-in-python/`, . Accessed: 2022-05-14.

[19] Support vector machines important questions. `https://medium.datadriveninvestor.com/support-vector-machines-important-questions-a47224692495`, . Accessed: 2022-05-14.

[20] Hyperparameter tuning - brief theory and what you won't find in the handbook. `https://medium.com/analytics-vidhya/hyperparameter-tuning-8ca311b16057`, . Accessed: 2022-05-14.

[21] Introduction to principal component analysis: Dimensionality reduction made easy. https://blog.bigml.com/2018/12/07/introduction-to-principal-component-analysis-dimensionality-reduction-made-easy/, . Accessed: 2022-07-14.

[22] Jean-François Lalonde, Nicolas Vandapel, Daniel F Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of field robotics*, 23(10):839–861, 2006.

[23] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[24] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[28] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008.

[29] Removing outliers using a conditional or radiusoutlier removal. https://pcl.readthedocs.io/projects/tutorials/en/latest/remove_outliers.html, . Accessed: 2022-07-14.

[30] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[31] Ji-Il Park, Jihyuk Park, and Kyung-Soo Kim. Fast and accurate desnowing algorithm for lidar point clouds. *IEEE Access*, 8:160202–160212, 2020.

[32] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A Chapman, Dongpu Cao, and Jonathan Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8): 3412–3432, 2020.

[33] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018.

[34] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[35] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[36] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[37] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized

spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.

[38] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[39] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[40] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[41] Hugues Thomas, François Goulette, Jean-Emmanuel Deschaud, Beatriz Marcotegui, and Yann LeGall. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. In *2018 International conference on 3D vision (3DV)*, pages 390–398. IEEE, 2018.

[42] Yiming Zeng, Yu Hu, Shice Liu, Jing Ye, Yinhe Han, Xiaowei Li, and Ninghui Sun. Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving. *IEEE Robotics and Automation Letters*, 3(4):3434–3440, 2018.

[43] Shang-Lin Yu, Thomas Westfechtel, Ryunosuke Hamada, Kazunori Ohno, and Satoshi Tadokoro. Vehicle detection and localization on bird's eye view elevation images using convolutional neural network. In *2017 IEEE International Sym-*

posium on Safety, Security and Rescue Robotics (SSRR), pages 102–109. IEEE, 2017.

[44] Nima Sedaghat, Mohammadreza Zolfaghari, Ehsan Amiri, and Thomas Brox. Orientation-boosted voxel nets for 3d object recognition. *arXiv preprint arXiv:1604.03351*, 2016.

[45] Liqiang Zhang and Liang Zhang. Deep learning-based classification and reconstruction of residential scenes from large-scale point clouds. *IEEE Transactions on Geoscience and Remote Sensing*, 56(4):1887–1897, 2017.

[46] Kai M Wurm, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. Improving robot navigation in structured outdoor environments by identifying vegetation from laser data. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1217–1222. IEEE, 2009.

[47] Benjamin Suger, Bastian Steder, and Wolfram Burgard. Terrain-adaptive obstacle detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3608–3613. IEEE, 2016.

[48] Leo Stanislas, Niko Suenderhauf, and Thierry Peynot. Lidar-based detection of airborne particles for robust robot perception. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA) 2018*, pages 1–8. Australian Robotics and Automation Association (ARAA), 2018.

[49] Nathan Sanchiz-Viel, Estelle Bretagne, El Mustapha Mouaddib, and Pascal Dassonvalle. Radiometric correction of laser scanning intensity data applied for terrestrial laser scanning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 172:1–16, 2021.

[50] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[51] Quang-Hieu Pham, Pierre Sevestre, Ramanpreet Singh Pahwa, Huijing Zhan, Chun Ho Pang, Yuda Chen, Armin Mustafa, Vijay Chandrasekhar, and Jie Lin. A* 3d dataset: Towards autonomous driving in challenging environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2267–2273. IEEE, 2020.

[52] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[53] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[54] Matthew Pitropov, Danson Evan Garcia, Jason Rebello, Michael Smart, Carlos Wang, Krzysztof Czarnecki, and Steven Waslander. Canadian adverse driving conditions dataset. *The International Journal of Robotics Research*, 40(4-5): 681–690, 2021.

[55] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceed-*

ings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2446–2454, 2020.

[56] Thierry Peynot, Steve Scheding, and Sami Terho. The marulan data sets: Multi-sensor perception in a natural environment with challenging conditions. *The International Journal of Robotics Research*, 29(13):1602–1607, 2010.

[57] Tensorflow. `https://www.tensorflow.org/`, . Accessed: 2022-07-16.

[58] Keras. `https://keras.io/`, . Accessed: 2022-07-16.

[59] scikit-learn. `https://scikit-learn.org/stable/`, . Accessed: 2022-07-16.

[60] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[61] Veldoyne library in ros. `http://wiki.ros.org/velodyne`, . Accessed: 2022-07-16.

[62] Point cloud library (pcl). `https://pointclouds.org/`, . Accessed: 2022-07-16.

[63] Eigen. `https://eigen.tuxfamily.org/index.php?title=Main_Page`, . Accessed: 2022-07-16.

[64] MATLAB. *version 9.11.0 (R2021b)*. The MathWorks Inc., Natick, Massachusetts, 2021.