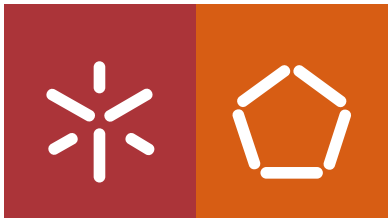
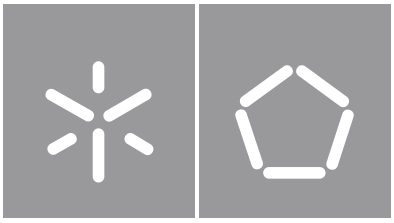


Universidade do Minho
Escola de Engenharia

André Agostinho Ribeiro Campos

ALFA-Pd: Hardware-assisted LiDAR Point Cloud Denoising

maio de 2022



Universidade do Minho
Escola de Engenharia

André Agostinho Ribeiro Campos

ALFA-Pd: Hardware-assisted LiDAR Point Cloud Denoising

Dissertação de Mestrado
Engenharia Eletrónica Industrial e Computadores
Sistemas Embebidos e Computadores

Trabalho efetuado sob a orientação de
Professor Doutor João Monteiro
Professor Doutor Tiago Gomes

maio de 2022

Agradecimentos

Em primeiro lugar, gostaria de agradecer às pessoas que sempre me apoiaram e tornaram possível tudo o que sou hoje. Aos meus pais e irmão, que sempre me apoiaram em todo o meu percurso, mesmo nos momentos mais difíceis. Aos meus orientadores, Professor Doutor Tiago Gomes e Mestre Ricardo Roriz, por todo o conhecimento transmitido, por toda a disponibilidade, e sobretudo pela confiança depositada em mim para a realização deste trabalho. Um sincero obrigado por todas as oportunidades e todas as lições.

Aos meus amigos de curso, que foram um grande apoio em todo o meu percurso académico. Um agradecimento especial para as pessoas com quem tenho o prazer de conviver e partilhar experiências diariamente – João Sousa, Luís Cunha, Pedro Sousa e Samuel Pereira e Mestre Diogo Costa. Um obrigado a todos que fizeram parte deste meu percurso de desenvolvimento, e que me deram a oportunidade de crescer e aprender.

Aos meu grupo amigos que sempre me apoiaram nos tempos mais difíceis. Um agradecimento particular a Ricardo Santelo, Bruno Justa, Eduardo Baptista, João Machado, Letícia Amorim, Manuel Raposo, Fábio Oliveira, Margarida Carvalho, Rita Vieira, Pedro Silva, Carlos Gomes e Tiago Amorim por toda a ajuda e momentos de amizade ao longo destes anos.

À restante família e todas as outras pessoas que me apoiaram durante todos estes anos: um sincero obrigado!

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Abstract

ALFA-Pd: Hardware-assisted LiDAR Point Cloud Denoising.

The interest in developing and deploying fully autonomous vehicles on our public roads has come to a full swing. As a result, interest in autonomous driving capabilities, already implemented in modern automobiles via Advanced Driver Assistance Systems (ADAS), is at an all-time high. However, employing highly reliable perception systems to navigate the environment is a very demanding task, requiring multiple sensors like Cameras, Radio Detection And Ranging (RADAR), and Light Detection And Ranging (LiDAR). The latter is critical for determining the distance and speed of surrounding obstacles while producing high-resolution 3D representations of the surrounding environment in real-time. Despite being assumed as a game-changer in the autonomous driving paradigm, LiDAR sensors can be susceptible to several noise sources, including internal components, mutual interference, light, and severe weather conditions. Notwithstanding the numerous mitigation approaches for weather denoising described in the literature, some issues still exist, ranging from high complexity to low accuracy or even systems with reduced overall performance. Thus, this dissertation proposes the ALFA-Pd framework, an embedded weather denoising approach that supports the existing and new state-of-the-art outlier removal methods using a reconfigurable hardware platform. ALFA-Pd contributes with a new weather denoising approach by combining the Dynamic Radius Outlier Removal (DROR) and the Low-Intensity Outlier Removal (LIOR) algorithms. The performed evaluation shows that Dynamic low-Intensity Outlier Removal (DIOR) can achieve better accuracy and performance while guaranteeing real-time requirements when compared to other state-of-the-art solutions.

Keywords: ADAS, Autonomous vehicles, LiDAR, FPGA, Hardware-accelerated, Point cloud, Point cloud filtering, Weather denoising.

Resumo

ALFA-Pd: Aceleração de métodos de redução de ruído em sistemas LiDAR.

A indústria automóvel representa um dos principais setores económicos a nível mundial. Esta indústria tem sofrido grandes evoluções tecnológicas nos últimos anos, especialmente no ramo de veículos de passageiros. Estas evoluções e a mudança de paradigma rumo à produção de veículos cada vez mais inteligentes, tem desafiado constantemente fabricantes e marcas a investir inúmeros recursos na pesquisa e desenvolvimento de soluções para assistir à condução autónoma. Este objetivo pode ser atingido recorrendo a um conjunto de sensores tais como *Light Detection And Ranging* (LiDAR), sendo estes últimos bastante utilizados em sistemas comerciais *Advanced Driver Assistance System* (ADAS). A utilização de sensores LiDAR em sistemas ADAS tem vindo a ganhar extrema importância, apresentando inúmeras vantagens em relação aos demais, tais como resolução, precisão, mapeamento 3D, funcionamento em ambientes com pouca luz, etc. Apesar das vantagens, o funcionamento de um sensor LiDAR pode ser severamente afetado devido a interferências provocadas por outras fontes luminosas, como por exemplo a luz solar, pela coexistência de outros sistemas LiDAR no mesmo raio de ação, ou pela influência de condições meteorológicas adversas. Apesar das numerosas abordagens de mitigação destes problemas na literatura, algumas questões ainda existem, desde alta complexidade até baixa precisão ou mesmo baixo desempenho em geral. Assim, esta dissertação propõe a *framework* ALFA-Pd, que oferece métodos de redução de ruído causado por condições climatéricas adversas existentes, apoiada por uma plataforma de hardware reconfigurável. Para além disso, ALFA-Pd contribui com um novo método de filtragem, combinando os algoritmos *Dynamic Radius Outlier Removal* (DROR) e *Low-Intensity Outlier Removal* (LIOR). Os testes e resultados obtidos permitem concluir que *Dynamic low-Intensity Outlier Removal* (DIOR) alcança melhor precisão e desempenho ao mesmo tempo que cumpre requisitos *real-time*.

Palavras-chave: Point Cloud, Ruído, Remoção de ruído, Hardware reconfigurável, FPGA, LiDAR, Condução autónoma.

Contents

List of Figures

List of Tables

Glossary

ADAS Advanced Driver Assistance Systems

ALFA Advanced LiDAR Framework for Automotive

AR Angular Resolution

AXI Advanced eXtensible Interface

BRAM Block RAM

CNN Convolutional Neural Network

CPU Central Process Unit

DARPA Defense Advanced Research Projects Agency

DDR Double Data Rate

DIOR Dynamic low-Intensity Outlier Removal

DROR Dynamic Radius Outlier Removal

DSP Digital Signal Processors

dToF direct Time of Flight

DVC Debugger Visualiser Configurator

FCSOR Fast Cluster Statistical Outlier Removal

FF Flip-Flop

FIFO First In First Out

FLANN Fast Library for Approximate Nearest Neighbors

FN False Negatives

FoV Field of View

FP False Positives

FPGA Field-Programmable Gate Array

FPS Frames Per Second

FPT Frame Processing Time

GCC GNU Compiler Collection

GUI Graphical User Interface

IP Intellectual Property

iToF indirect Time of Flight

LiDAR Light Detection And Ranging

LIOR Low-Intensity Outlier Removal

LRR Long-Range RADAR

LUT LookUp Table

MCU MicroController Unit

MPSoC MultiProcessor System on Chip

NF Neighbor Finder

NIR Near-InfraRed

PC Point Cluster

PCL Point Cloud Library

Pd Platform denoising

PL Programmable Logic

PR Points Removed

PS Processing System

RADAR RAdio Detection And Ranging

ROI Region Of Interest

ROR Radius Outlier Removal

ROS Robot Operation System

SAE Society of Automotive Engineers

SNR Signal-to-Noise Ratio

SOR Statistical Outlier Removal

SRR Short-Range RADAR

SVM Support Vector Machine

ToF Time of Flight

TP True Positives

VG Voxel-Grid

VTK Visualization ToolKit

1. Introduction

Nowadays, more than a decade after the first self-driving car winning the Defense Advanced Research Projects Agency (DARPA) Challenge, the interest in creating and deploying completely autonomous vehicles is at an all-time high. An autonomous vehicle requires trustworthy solutions to produce an accurate map of its surroundings, which is only possible through the use of multi-sensor perception systems equipped with RAdio Detection And Ranging (RADAR), Cameras, and Light Detection And Ranging (LiDAR) sensors [1, 2, 3, 4]. These multi-sensor perception systems empower vehicles with the ability to detect the distance and speed of nearby obstacles, as well as their shape, so they can drive safely through the environment, contributing to the various levels of driving automation defined by the Society of Automotive Engineers (SAE). Despite the use of LiDAR sensors in the automotive sector being relatively new, they are now widely regarded as a key technology for fully autonomous vehicles due to their ability to generate high-resolution 3D representations of their surroundings in real-time [5, 6, 7].

LiDAR technology is constantly evolving and thus finding different usages in a wide range of applications. Since accurate and exact measurements of the environment using a 3D point cloud representation can aid perception systems [3], they are used in a variety of applications, including obstacle identification, object and vehicle detection [8], pedestrian recognition and tracking [9], and ground segmentation for road filtering [10, 11]. Nonetheless, a variety of noise sources, including internal components [12], mutual interference [13, 14], reflectivity issues [15], light [5], and severe weather conditions [4, 16, 17], can corrupt LiDAR's sparse 3D point clouds, difficulting the task of understanding the vehicle's surroundings.

Most algorithms are designed to operate under ideal weather conditions or ignore its impact on the sensor output. However, adverse conditions can reduce the vehicles' perception system performance, which can result in unexpected behavior. Recently, this issue has been extensively addressed in the literature [4, 16, 17, 18], with multiple studies benchmarking the sensor's responsiveness in a variety of diverse and harsh weather conditions, including fog [19, 20, 21, 22, 23], rain [19, 20, 24, 25, 26, 27], and snow [28, 29]. Solutions aiming to address the issue can be grouped as: (i) simulators for analyzing the impact of adverse weather on various road conditions and scenarios [22, 24, 25, 26, 27]; (ii) filters to improve background filtering and object clustering methods for processing roadside LiDAR data [23, 29]; (iii) denoising Convolutional Neural Network (CNN)-based learning approaches [20, 22]; and (iv) weather classification systems [17, 19, 23], which uses LiDAR data to forecast weather and change the sensor's operation based on changes in the atmosphere and asphalt, which leads to a better mapping of the

surroundings in real-time.

1.1 Main Goals

This dissertation aims to reduce the LiDAR point cloud deterioration induced by adverse weather conditions by introducing Advanced LiDAR Framework for Automotive (ALFA)-Pd, a framework that can execute current state-of-art algorithms. Moreover, to help mitigate the current performance limitations of these algorithms, ALFA-Pd offers hardware-accelerated capabilities to further improve state-of-art algorithms that present the best overall true positive per false positive ratios. Therefore, to successfully deploy ALFA-Pd, it must comply with the following requirements:

1. ALFA-Pd must be capable of selecting and executing different denoising methods at run time, which enables the selection of an appropriate filter to a specific weather condition.

2. ALFA-Pd must have hardware-accelerated capabilities to enable real-time filtering without introducing delays in the executing system.

3. ALFA-Pd must use current 3D LiDAR point cloud standards to facilitate the integration with high-level applications. Additionally, to ensure interoperability with other systems, ALFA-Pd must have a Robot Operation System (ROS) interface, as the majority of native LiDAR sensor drivers support ROS.

4. ALFA-Pd must study new weather denoising approaches that can benefit from existing algorithms. Viable approaches must be supported with or without the hardware acceleration tools, providing the best possible performance while maintaining platform flexibility.

1.2 Document Structure

This dissertation is structured as follows: Chapter 2 provides an overview of LiDAR technology and its applications in the automotive sector. Then, current issues posed by applying LiDAR sensors in automotive solutions are briefly exposed, narrowing into LiDAR point cloud corruption due to weather-related noise since it is the main topic of this dissertation. The Chapter concludes with a comprehensive analysis of current solutions and the presentation of a novel technique that improves over current state-of-art methods. Chapter 3 describes the platforms and tools used to develop this dissertation, as well as their operation. Afterwards, Chapter 4 gives an overview of the ALFA-Pd framework, its implementation modules and a working description of ALFA-DVC, a tool developed in this dissertation that enables real-time point cloud visualization, filter configuration, and debugging. Next, Chapter 5 offers an in-depth analysis of the performance of state-of-art weather denoising algorithms, including their hardware-accelerated implementation. Finally, Chapter 6 concludes the work developed throughout this dissertation and proposes future improvements.

2. Background and State of the Art

This Chapter starts with an analysis regarding the main sensors used for autonomous driving. Then, in Section 2.2, LiDAR technology is examined in detail, alongside its applications and challenges. Afterwards, Section 2.3 surveys current state-of-art solutions for LiDAR challenges generated by adverse weather conditions. Finally, Section 2.4 introduces a novel approach for LiDAR weather denoising based on existing methods.

2.1 Automotive Perception Sensors

An autonomous vehicle must recognize and understand the world around it to safely navigate its surroundings. Therefore, the majority of autonomous cars employ multi-sensor perception systems that typically include RADAR, Cameras, and LiDAR sensors [1, 2, 3, 4]. Each of these sensors offers several advantages and disadvantages. Nevertheless, they are capable of creating an accurate recreation of the real world when combined [7].

2.1.1 RADAR

The RADAR sensors are the most commonly used sensors in automotive detection and tracking applications [30]. It uses electromagnetic radio waves, to determine an objects' range, direction, and velocity [31], and depending on the measuring range, RADAR sensors are classified as Short-Range RADAR (SRR) and Long-Range RADAR (LRR) [32]. SRR sensors provide accurate range measurements in a broader Field of View (FoV), enabling applications such as blind-spot detection [33, 34], parking assistance, obstacle recognition [35, 36], and collision avoidance [30, 31]. However, they cannot achieve the high resolutions offered by LRR sensors because SRR sensors operate at a low frequency, typically 24 GHz. On the other hand, LRR sensors operate at a higher frequency, typically 77 GHz, thus offering accurate distance measurements [37]. Therefore, LRR sensors are frequently employed in adaptive cruise control applications [38, 39], where detection of objects at great distances is critical. Nonetheless, they require more complex antenna systems, which raises their production cost.

2.1.2 Camera

From photos to video, Cameras are the most accurate way to create a visual representation of the world, a highly required feature in self-driving cars. Most autonomous vehicles rely on Cameras mounted on all four vehicle sides to create a 360 degrees view of the surrounding environment [40]. To accomplish this goal, some use Cameras with a wide FoV, up to 120 degrees, and with a limited range [41], while others provide long-range visuals with narrow FoV [42]. Although Cameras provide accurate views, they have limitations in low-visibility environments such as fog, rain, or night [43]. Additionally, Cameras cannot measure the depth and distance to an object, as it must be estimated through calculations [6].

2.1.3 LiDAR

LiDAR sensors work by targeting an object with light pulses and measuring the time for the light to return. This sensor, similar to Cameras, can provide high resolution representations of the surroundings. However, instead of outputting 2D images, which Cameras provide, LiDAR sensors output images in a point cloud (2D or 3D) format. These representations enable autonomous vehicles to read their surroundings, fulfilling the requirements for applications like object detection and classification [7]. Similar to RADAR and in contrast with Cameras, LiDAR sensors do not require external light sources because it features active illumination, which mitigates the effects of environmental conditions.

2.2 Automotive LiDAR

Obtaining distances through measuring travel-time and intensity of light beams date back to the pre-laser decade (1930s) [44, 45], but it was only in 1953 that the concept of LiDAR appeared [46, 47]. The LiDAR principle has remained unchanged since then, as a method of measuring distances by calculating the round-trip time of a flight pulse traveling between the sensor and a target.

2.2.1 LiDAR Working Principle

Typically, LiDAR sensors operate by emitting light into their FoV through one or more laser beams. Some LiDAR transmitters use amplitude modulated laser pulses, in the spectrum's Near-InfraRed (NIR) region, which allow for better Signal-to-Noise Ratio (SNR) [7]. Subsequently, the receiver gathers the reflected light, organizes the received data, and computes the acquired information to build a point cloud.

Measurement Principles

LiDAR sensors can measure the distance to a target using one of two methods: direct Time of Flight (dToF) or indirect Time of Flight (iToF). The dToF principle employs a simpler architecture providing a

cheaper implementation [7]. However, dToF is less resistant to mutual interference from other LiDAR sensors. In comparison, iToF is a more robust technique that produces better results, but it is substantially more complex and thus requires higher production costs [6].

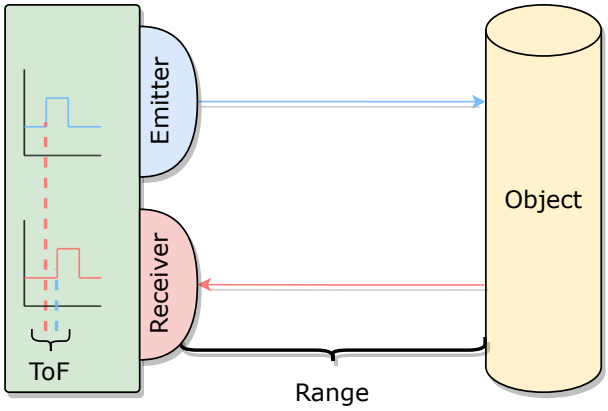


Figure 2.1: LiDAR working principle.

The dToF technique determines the Time of Flight (ToF) by calculating the time required for light to travel from the sensor to an object and back, as depicted in Figure 2.1. This method offers several advantages because it does not take the phase of the signal into account. Another important factor is that dToF techniques are far more affordable than other solutions. However, since the emitter and receiver are simple, measuring ToF is less accurate, making it susceptible to external influences such as external light sources and weather conditions, which results in a lower SNR.

In contrast, iToF is a more sophisticated approach. It is the most advanced silicon-based ToF technology on the market, with numerous commercial implementations. Nonetheless, the measurement range is significantly lower because longer ranges require longer pulses, which reduces the SNR. Additionally, multi-object disambiguation can occur in iToF approaches as the estimated distance is an average of all sources of optical reflection [48]. Moreover, SNR can be further degraded because complementary metal-oxide semiconductor detectors often lack signal gain, have a low fill factor, and have lower responsivity than the ordinary photodiodes used in dToF.

Imaging Techniques

Numerous techniques were developed and implemented in LiDAR sensors to create 3D point clouds to recreate the real world. These techniques are divided into two broad categories: those that employ beam-steering mechanisms to scatter a light signal across the environment spot, and those that illuminate the entire FoV simultaneously.

LiDAR systems with beam-steering mechanisms are often divided into rotor-based mechanical LiDAR systems and scanning solid-state LiDAR systems [49]. The rotor-based systems are the most mature scanning technique, having a strong presence in current automotive perception systems [7]. Figure 2.2

Figure 2.2: Velodyne VLP-16.

depicts a rotor-based LiDAR sensor. To achieve 360° horizontal detection, they employ a mechanical rotor that spins the emitter/receiver pairs. However, since they contain moving parts, the frame rate is limited by the sensor rotation speed, influenced by the friction between components, resulting in a performance decrease. Thus, current rotor-based LiDAR systems only generate data at a rate between 10 Hz to 20 Hz.

Scanning LiDAR systems focus light in a beam to illuminate the target, as depicted in Figure 2.3. Additionally, these systems use a low divergence laser with some form of beam-steering as its emitter, which defines the vertical and horizontal resolution. These techniques enable longer-range scanning by relying solely on the system's optics, resulting in a highly adaptable design. However, because the FoV points do not share a common light source, these systems become susceptible to external interference, reducing the SNR. Additionally, these beam guiding technologies degrade frame rate, operating at a fraction of the speed offered by flash LiDARs.

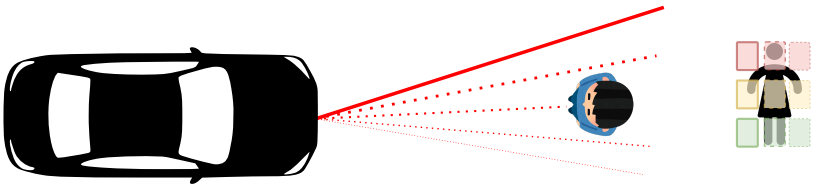


Figure 2.3: Scanning LiDAR.

Scanning solid-state LiDAR systems are another LiDAR beam-steering mechanism, represented by one example depicted in Figure 2.4. Since they lack a rotor mechanism that spins the emitter/receiver pairs, these systems do not reach 360 degrees of horizontal FoV [50]. However, by having no mechanical parts spinning, there is no friction between components, which results in the scanning solid-state LiDAR systems being capable of achieving greater frame rates when compared to mechanical ones. Additionally, their simplicity makes them cheaper to produce. Due to their scalability in the automotive industry, they are frequently employed in groups of sensors to increase the FoV.

Since Flash LiDAR systems have no moving parts, they are often named as complete solid-state sensors. Similar to traditional digital cameras, these sensors use a flash to illuminate the environment and

Figure 2.4: Velodyne Vellaray H800.

photodetectors to capture the reflected light, as illustrated in Figure 2.5. Additionally, these systems are immune to light distortion, as all FoV points share the same light source. However, due to power constraints, the measuring range of Flash LiDAR sensors is limited, as the only way to increase the measurement range is by increasing the laser's power.

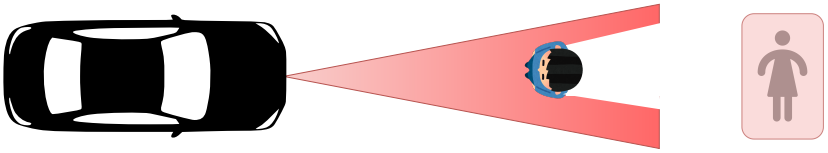


Figure 2.5: Flash LiDAR.

2.2.2 LiDAR Applications

Advanced Driver Assistance Systems (ADAS) are designed to help drivers to avoid collisions by stopping the vehicle or conducting evasive maneuvers. Therefore, these systems must constantly scan the vehicles' surroundings for potentially harmful circumstances [51]. For instance, Wei et al. [52] developed an ADAS based on LiDAR sensors to avoid collisions by automatically braking the vehicle in unsafe situations.

In recent years, the research on autonomous vehicles has gained considerable attention, with pedestrian recognition and tracking ranking among the most critical topics [53, 54]. Nevertheless, to predict the possibility of a future collision, pedestrian recognition systems have to accurately recognize pedestrians as far away as possible, which is enabled by using LiDAR technology [36]. Therefore, numerous methodologies such as stochastic optimization [55, 56], doppler effect [32, 57], and clustering [36] were developed. Additionally, Kidono et al. [58] proposed a method of pedestrian recognition based on Support Vector Machine (SVM) techniques, where is shown the advantages and disadvantages of using LiDAR sensors over other sensors.

2.2.3 LiDAR Challenges

According to Hasirlioglu et al. [53], over 1.2 million people die in traffic accidents each year, highlighting the critical need for some form of driver assistance to prevent more human deaths. For this, it is vital to collect trustworthy information about the environment, allowing these systems to actively avoid dangerous

scenarios. However, the sparse 3D point clouds produced by an automotive LiDAR sensor can be corrupted by a variety of sources of noise, such as internal components [12], mutual interference [13, 14], reflectivity issues [15], light [5], adverse weather conditions [4, 16, 17], among others [18].

Weather Influence

One of the major drawbacks of LiDAR technology is its susceptibility to harsh weather, as it can degrade the performance by up to 25% [53]. However, extreme weather conditions are a vast subject, and each weather condition, such as fog, rain, or snow, has a unique effect on the LiDAR output. Firstly, fog is characterized by Hasirlioglu et al. [53] as a multifaceted phenomenon influenced by various elements, including droplet microphysics, aerosol chemistry, radiation, turbulence, large/small-scale dynamics, and droplet surface conditions. In summary, fog is a collection of tiny water droplets with diameters less than 100 mm suspended in the air. Additionally, it is important to note that fog behaves differently at sea, in the atmosphere, and on the ground. For example, the fog droplet size on the ground is smaller if compared with droplets in cumulus clouds. Nonetheless, analyzing and studying every fog scenario is far too time-consuming for any practical application, so Hasirlioglu et al. [59] developed a fog simulation to accelerate the process of evaluating LiDAR performance. Wherefore, with the data provided by the author, it is possible to deduce that fog has a detrimental influence on wave propagation, which is further disrupted as fog density increases.

Heinzler et al. [19] published an in-depth analysis of vehicle LiDAR sensor performance under extreme weather circumstances such as heavy rain and dense fog. The author finds that the quality and precision of measurements are likely to be compromised due to atmospherics particles scattering too much laser power. Moreover, in deep fog, where visibility is limited to 20-40 meters, almost all primary laser returns are absorbed in a range of less than five meters, indicating that the absorptions were created by fog. Nonetheless, highly reflective objects such as taillight retroreflectors are unaffected in the same conditions.

By constructing a mathematical model for the performance degradation of LiDAR as a function of rain rate, Goodin et al. [24] conducted quantitative research on how rain rate affects ADAS performance. Later in the article, this model is integrated into a simulation of an obstacle-detection system to demonstrate how it may be used to forecast the rain effect on LiDAR measurements statistically. The author concludes that the increasing rain rate significantly affects the created point cloud. However, the reduction in LiDAR range has a negligible effect on obstacles' distance detection, indicating that the sensor's capacity to identify obstacles is not significantly affected by rain.

2.3 Point Cloud Weather Denoising Methods

As previously stated, the majority of current data processing algorithms operate under ideal weather conditions. Nonetheless, adverse conditions can affect the regular operation of the perception system.

[4, 16, 17, 18], making it critical to develop solutions that overcome present constraints in terms of performance, accuracy, and overall system complexity. However, LiDAR point clouds are impacted differently depending on the nature of the weather conditions. While rain and fog contain water droplets that scatter the emitted light, decreasing the effective working range and resulting in wrong measurements, smoke and snow contain solid particles that can generate ghost information in the point cloud.

The LiDAR noise generated by harsh weather conditions often manifests itself in the form of outliers. In a defined dataset, observations that deviate significantly from the adjacent others are considered outliers. On a 3D point cloud data, outliers are points that share no correlation or attributes with their neighbors regarding distance or intensity. These outlier points are primarily associated with noise, and removing them from the point cloud allows high-level applications such as object detection algorithms to achieve higher accuracy ratios. Therefore, several approaches were developed to eliminate outliers caused by weather conditions in LiDAR systems. These approaches include outlier removal techniques and machine learning techniques. Aside from outlier removal methods, learning based denoising algorithms also started to emerge [20, 22]. They are, nonetheless, considered complex since they require the use of real-world datasets and powerful computational resources. As a result, they fall outside the scope of this dissertation.

2.3.1 Voxel-Grid Filter

Voxel-Grid (VG) [60] filter, depicted in Figure 2.6, consists in defining 3D boxes (forming a voxel grid) in the 3D space of the point cloud. Then, for each voxel, the algorithm selects a point (usually the central point or the centroid of the box) to approximate the remaining points inside the voxel. Because of this feature, VG filters can be used for point cloud denoising since a noise point often lacks in neighbors or does not share information with them, which will end in being removed from the point cloud. VG methods are fast and relatively simple to implement. However, because they cause the down-sampling of the information within the voxel, not only noise points will be removed from the point cloud but also points with useful information about the surroundings.

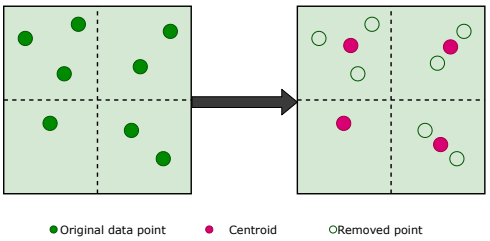


Figure 2.6: VG working principle.

2.3.2 Radius Outlier Removal

The Radius Outlier Removal (ROR) algorithm uses a k-d tree data structure to compute each point's mean distance to its neighbors within a user-defined radius **R1**, as seen in Figure 2.7. If the number

of neighbors inside the chosen radius is less than the user-defined threshold, the point is considered an outlier and is removed from the point cloud. Therefore, the performance of this filter is highly dependent on the radius and the minimum number of chosen neighbors. While this filter has the advantage of being simple to implement, the fixed filter radius search becomes problematic when applied to sparse 3D LiDAR sensors. As the detection range rises, the space between points increases as well due to the horizontal and vertical resolution of LiDAR systems. Consequently, this filter will most likely eliminate points collected at long distances.

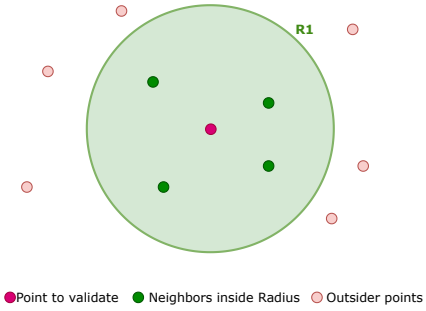


Figure 2.7: ROR working principle.

2.3.3 Statistical Outlier Removal

Statistical Outlier Removal (SOR) is a denoising method that removes the outlier points based on neighbor information (Figure 2.8). However, instead of using a fixed radius and a minimum threshold for the number of neighbors, first it calculates the average distance **R1** of each point to its neighbors, defined as "k-nearest neighbor", rejecting the points whose distance is higher than **R2**, i.e., the average value plus the standard deviation. Despite improving ROR in detecting outlier points, SOR severely increases the computation overhead.

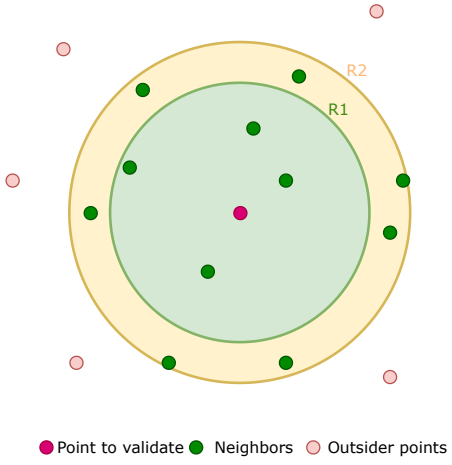


Figure 2.8: SOR working principle.

2.3.4 Fast Cluster Statistical Outlier Removal

The Fast Cluster Statistical Outlier Removal (FCSOR) [61] filter is a merge between VG and SOR filters that aims at improving the time performance of SOR. Before calculating the distance to neighbors, FCSOR performs a sub-sampling of the point cloud with a VG filter step (Figure 2.6). However, and despite decreasing the computational complexity due to the reduction of the number of points, it still does not fit the real-time requirements, and the success rate in detecting outlier points slightly decreases.

2.3.5 Dynamic Radius Outlier Removal

The Dynamic Radius Outlier Removal (DROR) [28] filter was developed to address the accuracy issues presented by SOR and ROR in sparse 3D LiDAR point clouds. Nonetheless, unlike the ROR filter, which has a fixed radius, the DROR filter has a dynamic radius. This radius is calculated by multiplying the sensor Angular Resolution (AR) by the point distance to the sensors illustrated in Figure 2.9. Furthermore, with the addition of the dynamically calculated search radius to ROR, DROR minimizes the wrong classification of distant points as outliers. According to the author's results, the DROR filter outperforms conventional filters, and when compared to the conventional ROR, it improves accuracy by more than 90%. However, despite its excellent accuracy, it suffers from performance concerns due to its high computational cost.

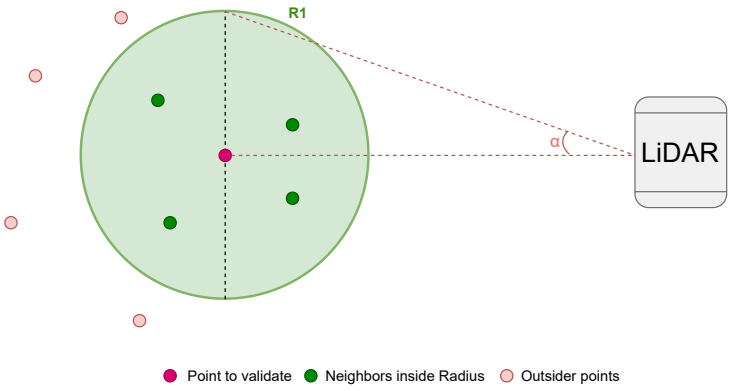


Figure 2.9: DROR working principle.

2.3.6 Low-Intensity Outlier Removal

Low-Intensity Outlier Removal (LIOR) is a method proposed by Park et al. [62] that aims at improving the speed and accuracy performance limitations of previous methods by removing the noise caused by snow or rain based on the intensity of the reflected light. Noise points usually present a lower intensity value when compared with neighbors at the same distance. Thus, every point below a defined threshold value is classified as an outlier. To reduce the false positive ratio, a second step is applied to each outlier, which can be turned into an inlier if several neighbors (defined by a threshold) are detected within a specified

distance. The working principle behind LIOR is based on the ROR algorithm (depicted by Figure 2.7) with the addition of the point intensity information. When comparing LIOR with the previous filtering methods, it can achieve filtering speeds up to 12x faster than SOR, and 8x faster than DROR. However, real-time filtering in high-speed vehicles is only possible if the method is applied only to certain Region Of Interest (ROI) rather than the full point cloud. Regarding the accuracy, the noise points can be filtered with the same efficiency as the DROR method. In their evaluation, LIOR claims to achieve a false positive ratio of 1%, while DROR reached almost 50% of points wrongly classified as outliers.

2.3.7 Discussion

By analyzing the weather denoising 3D point clouds state-of-art algorithms, it is possible to conclude that there is not one-size-fits-all algorithm. Firstly, the VG filter can achieve real-time speeds by downsampling point clouds, which results in a weak denoising effect. The SOR and ROR filters are the most mature, having been used in a wide variety of LiDAR applications. However, they present bad denoising performance due to the sparsity of point clouds, making them unviable for autonomous applications. FCSOR was developed to enhance SOR's runtime performance by parallelizing and adding a voxel step, but it has a low true positive ratio. Finally, DROR and LIOR can be regarded as state-of-the-art algorithms since they are the most accurate and suitable for 3D point clouds.

Table 2.1: Weather denoising algorithms summary.

Algorithm	Low complexity	Sparsity ready	Dynamic radius	Intensity based	Good TP ratio	Real-time
VG	X					X
SOR	X					
FCSOR						X
ROR	X					
DROR	X	X	X		X	
LIOR	X			X		X

The Table 2.1 summarizes the denoising methods analyzed and reproduced by this dissertation, highlighting the benefits and drawbacks of the state-of-the-art techniques. DROR and LIOR are the ones with the best results. The DROR author asserts that it is capable of attaining a high true positive rate at the expense of considerable processing resources. On the other hand, LIOR author states that the filter is capable of operating in real-time by classifying points based on their intensity levels. However, there is a need for an algorithm that can attain the high true positive ratio of DROR while still complying to the time constraints imposed by LIOR.

2.4 Dynamic low-Intensity Outlier Removal

Dynamic low-Intensity Outlier Removal (DIOR) is a novel approach proposed by this dissertation [63]. DIOR uses the LIOR concept as its core, in which it classifies points as inliers or outliers based on their

intensity values. However, instead of using a ROR based technique to validate points as outliers, DIOR uses the DROR methodology of using a dynamically calculated search radius (depicted in Figure 2.10). This search radius is calculated by multiplying the LiDAR sensor's horizontal resolution to the distance of a given point. Therefore, DIOR is ready for the sparsity of 3D point clouds because the search radius increases as the distance between points grows, resulting in DIOR outperforming DROR in terms of speed performance while keeping DROR's high true positive ratio. Since DIOR was developed in this dissertation, the full explanation of this algorithm is further explained in the Chapter 4.

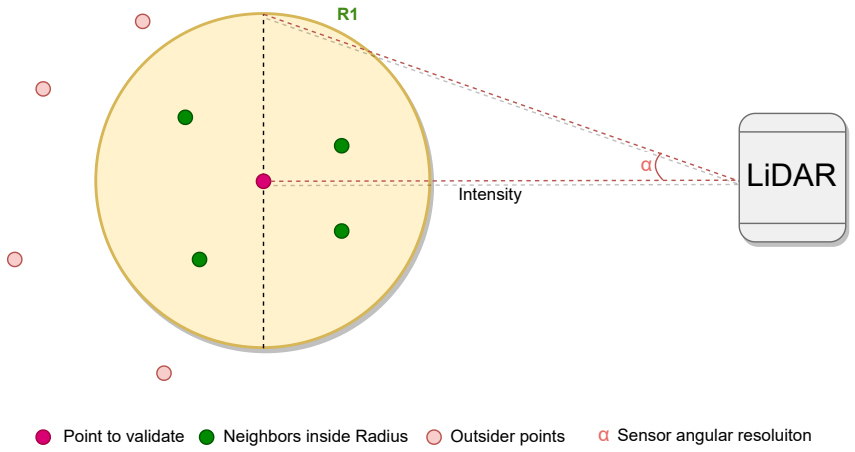


Figure 2.10: DROR working principle.

3. Platform and Tools

This Chapter specifies the platform and tools that were used throughout the development of this work. Since the framework must support both software and hardware tools, Section 3.1 will explain the platform and why it was chosen, while Sections 3.2 and 3.3 will go into the tools used.

3.1 Reconfigurable Technology

Field-Programmable Gate Array (FPGA) technology enables the development of custom hardware accelerators by providing flexibility and functionality with customizable hardware blocks. FPGAs' hardware fabric works by having an array of programmable logic blocks and a hierarchy of reconfigurable interconnects that allow them to be linked together, much like a large number of logic gates that may be arranged in a variety of different configurations. Additionally, using an MicroController Unit (MCU) allows the deployment of higher-level applications that can control/configure the hardware layer in real-time and run more complex parts of the algorithms, easing the implementation and deployment process. Therefore, to fulfill the framework's hardware-software co-design, a platform featuring FPGA resources and an MCU is needed.

Zynq UltraScale+ MPSoC ZCU104

The selected platform to run all of the functionalities developed in this dissertation was the Zynq UltraScale+ MultiProcessor System on Chip (MPSoC) ZCU104 [64], which includes the XCZU7EV-2FFVC1156 MPSoC for hardware development. This MPSoC enables designing embedded applications such as ADAS with the support of video codecs, while having plenty of peripherals and interfaces for embedded video solutions. The MPSoC's Processing System (PS) is comprised of a quad-core Arm Cortex-A53 application processor running at 1.2 GHz, a dual-core Cortex-R5 real-time processor, a Mali-400 MP2 graphics processing unit, a 4KP60 capable H.264/H.265 video codec, and 2 GB of Double Data Rate (DDR)4 memory running at 525 MHz. Regarding the FPGA, it provides 23040 LookUp Table (LUT), 46080 Flip-Flop (FF), 312 Block RAM (BRAM)s, and 1728 Digital Signal Processors (DSP).

3.2 Robot Operating System

ROS [65] is a collection of tools, libraries, and conventions aimed at simplifying the work of developing complex and robust robot behavior across a broad range of robotic platforms. It provides services for heterogeneous computer clusters and the implementation of messages between processes and devices connected to the same network. ROS processes are represented as nodes in a topic-based graph structure. Furthermore, ROS nodes interact, respond to service requests, and provide services to other nodes. Additionally, they can obtain and modify configuration data through a shared database known as the parameter server. All of this is made possible by a process called the ROS Master, which registers nodes, establishes and manages node communication. Rather than routing messages and service calls through the master, ROS establishes peer-to-peer communication between all node processes upon their registrations. This decentralized design is well-suited for autonomous applications, frequently composed of networked computer hardware and that may interface with off-board computers for computationally intensive tasks or directives. Due to its reliance on a wide array of open-source software, ROS runs on top of a Linux system, easing the communication of ALFA-Platform denoising (Pd) to numerous LiDAR sensor drivers. By publishing and subscribing to ROS topics, the ROS environment also enables the development of a tool to configure and visualize the output of the ALFA-Pd platform.

3.3 Point Cloud Library

The Point Cloud Library (PCL) [66] is an open-source library of algorithms for processing 2D and 3D point clouds. It includes algorithms for filtering, feature estimation, surface reconstruction, 3D registration, model fitting, object recognition, and segmentation. In this work, the PCL algorithms are used to implement/deploy the weather denoising methods described in the state-of-art. PCL also offers different formats to use and store point clouds: the Point Cloud Data (.pcd) and the Polygon File Format (.ply), which enables the usage of publicly available datasets in this dissertation.

Additionally, PCL is accessible on most operating systems. It is written in C++ and is fully integrated with ROS, allowing it to run on embedded systems such as the one used to deploy this dissertation's work. It is critical to note that this library depends on several third-party libraries to operate properly, including the Eigen library, primarily used for mathematical operations, the Visualization ToolKit (VTK), enabling point cloud visualization, Boost for shared pointers, and the Fast Library for Approximate Nearest Neighbors (FLANN) to support fast k-nearest neighbor search.

3.4 Qt

Qt [67] is a software development toolkit for creating cross-platform Graphical User Interface (GUI) that runs on desktop, mobile, and embedded platforms and is compatible with a variety of compilers,

including the GNU Compiler Collection (GCC) C++ compiler. Qt is used in this dissertation to construct the ALFA-DVC which is a tool that was developed from scratch that allows real-time point cloud visualization debugging and configuration of all the parameters of the embedded system's software/hardware.

3.5 OpenEmbedded

OpenEmbedded [68] is a framework for automating the build process and a cross-compilation environment of developing embedded Linux distributions. OpenEmbedded is the recommended build method for the Yocto Project [69], the Linux Foundation workgroup that develops Linux-based embedded systems. Additionally, OpenEmbedded recipe collections are organized in layers, with the lowest layer containing platform- and distribution-independent metadata. OpenEmbedded is employed in this dissertation to generate the Linux image with ROS, PCL, and all the essential packages. Furthermore, OpenEmbedded is used to cross-compile all the software by invoking a custom recipe that fetches the code from GitHub and installs it into the ROS environment.

3.6 Advanced LiDAR Framework for Automotive

ALFA is an in-house open-source framework for automotive that aims to offer a multitude of helpful features for the validation and development of LiDAR-based solutions. These features include: (i) Generic and multi-sensor interface; (ii) Pre-processing algorithms for data compression, noise filtering, ground segmentation, amidst others; (iii) Configurable output for high-level applications; and (iv) Reconfigurable point-cloud representation architecture.

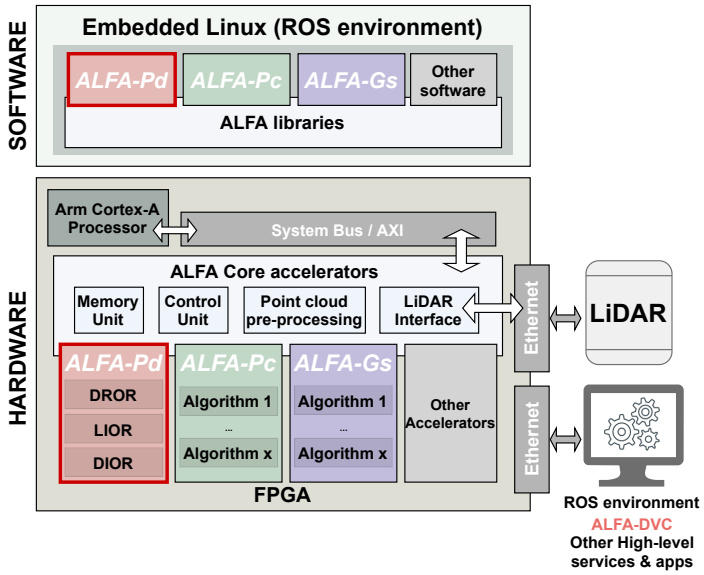


Figure 3.1: ALFA architecture block diagram.

ALFA features a modular design, where its blocks are totally independent from each other, which enhances the framework's flexibility and functionality. These blocks can be divided into two types: Units and Extensions. The core components that provide resource management, control, and interface are defined as ALFA Units and together form the ALFA Core, being the former where this dissertation is included, as depicted in Figure 3.1. Developers can build algorithms for specific domains on top of the core, hence expanding ALFA's capabilities. By avoiding the need to implement a whole system to evaluate an algorithm or a subset of it, ALFA significantly reduces the time required to test and design new LiDAR-based algorithms. While the framework presented in this dissertation can be used independently, by integrating with ALFA, it is feasible to access point cloud data without incurring the overhead of PS-Programmable Logic (PL) communication, hence enhancing the speed of the developed algorithms even further.

4. ALFA-Pd Implementation

The ALFA-Pd framework enables real-time point cloud processing for automotive applications, by offering state-of-art weather denoising methods with built-in hardware-accelerated capabilities. Figure 4.1 depicts the overall architecture of the ALFA-Pd framework. On the embedded platform, the framework supports a collection of software-based denoising methods (assisted by the PCL), and core-libraries that interface with the weather denoising accelerators. It features a software layer, which uses a ROS environment on top of a minimalist embedded Linux, providing different levels of abstraction for high-level applications. On the other hand, the hardware layer features several core blocks such as a hardware interface for the software libraries; a memory management module; and distance calculator units to be used by the hardware weather denoising accelerators. Additionally, the framework supports a high-level application, ALFA-DVC. This tool enables point cloud visualization, weather denoising debugging features, and can configure the embedded platform.

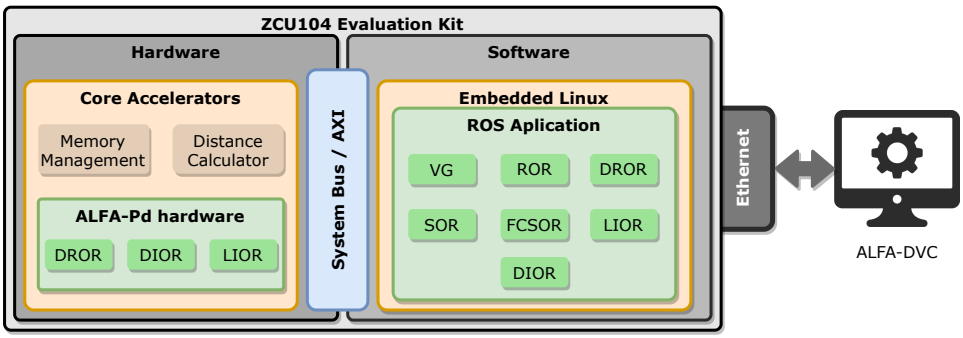


Figure 4.1: ALFA-Pd framework architecture.

Currently, ALFA-Pd supports the deployment and evaluation of the following state-of-the-art denoising algorithms: VG, SOR, FCSOR, ROR, DROR, LIOR, and the new method developed in this dissertation, DIOR. At runtime, the user can select the denoising algorithm to employ and the type of system used to execute it (software-only or hardware-accelerated). Regarding the software versions, the filter parameters can be changed in real-time, alongside execution parameters such as thread count. ALFA-Pd uses ROS messages to receive noisy point cloud data, transmit its denoised version, and system configuration.

As depicted in Figure 4.2, the ALFA-Pd embedded platform is divided into three main modules: software, memory, and hardware. Firstly, the software module receives the noisy point cloud data through a ROS topic, and, depending on the configuration, the point cloud is filtered in the software or hardware

layers. After the filtering process, the software module publishes the filtered point cloud to a ROS topic alongside the filter performance metrics.

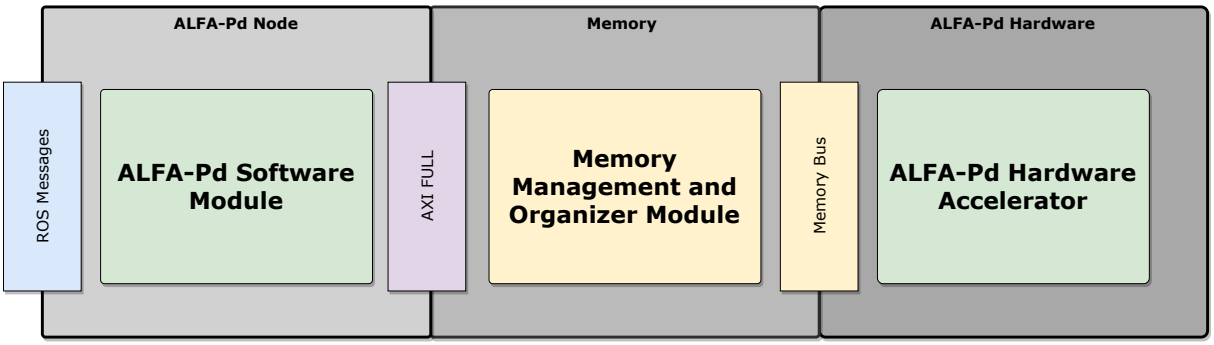


Figure 4.2: ALFA-Pd modules overview.

The memory module is responsible for interfacing the software and hardware modules using the Advanced eXtensible Interface (AXI)-Full protocol. There are two different types of memory implementations present in the platform. The BRAM memory implementation provides faster execution times, while the DDR implementation features a more generic implementation, offering more portability. Despite using the AXI-Full protocol to communicate with the software layer, they feature distinct communication protocols with the accelerators for the weather denoising algorithms. If the BRAM memory implementation is in use, the hardware modules wait a start signal from the PS to start executing. Then, the filtered point cloud is stored inside a specified position in the BRAM, accessible by the software. On the other hand, the point cloud data is accessed using AXI-Full transactions when using the DDR version. Once the hardware modules finish processing a frame, the outliers are removed from the memory.

4.1 ALFA-Pd Software

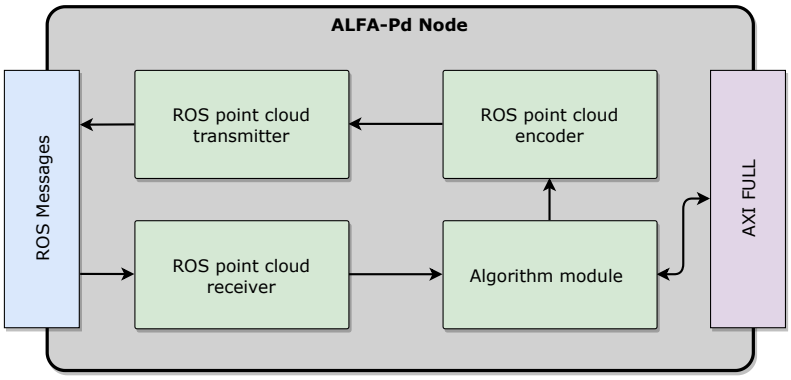


Figure 4.3: ALFA-Pd software modules architecture.

The ALFA-Pd software layer is supported by the ALFA-Pd ROS node. It is divided into several modules, as depicted in Figure 4.3. The first module to execute is the *ROS point cloud receiver*, responsible for

receiving point cloud data and transforming it into the PCL format. The *ROS point cloud encoder* is responsible for converting the point cloud to the ROS format, later published into a ROS topic by the *ROS point cloud transmitter* module. Moreover, this module receives the configuration settings to apply to the filters after. Lastly, the *Algorithm module* handles point cloud denoising tasks. It receives the data from the previous module and executes one filter depending on the configuration. Most filters are implemented using the PCL library, which provides several point cloud processing functionalities such as neighbor search. Table 4.1 lists the denoising algorithms alongside the PCL functions used to implement them. The PCL library allows for straightforward implementations of the ROR, SOR, and VG algorithms. The FCSOR, LIOR, and DROR were implemented using the algorithms provided by their authors.

Table 4.1: PCL software modules used by the ALFA-Pd and the ALFA-DVC tool.

Algorithm	PCL module(s)	Description
Voxel Grid	**VoxelGrid()**	Assembles a local 3D grid over a given PointCloud, and downsamples plus filters the data.
ROR	**RadiusOutlierRemoval()**	Filters points in a cloud based on the number of neighbors they have.
SOR	**StatisticalOutlierRemoval()**	Uses point neighborhood statistics to filter outlier data.
FCSOR	**StatisticalOutlierRemoval()** **VoxelGrid()**	These two PCL modules were combined to implement the FCSOR algorithm.
DROR	**kdtree.radiusSearch()**	Obtains the number of neighbors inside a dynamic radius R1.
LIOR	**kdtree.radiusSearch()**	Obtains the number of neighbors inside a fixed radius.
DIOR	**kdtree.radiusSearch()**	Obtains the number of neighbors inside a dynamic radius R1.

Multithreading Configuration

Most of the state-of-art algorithms are designed to run with single-thread configurations. Therefore, since ALFA-Pd is designed to run in embedded systems with limited computational power, multithreading was employed to boost the system's performance. The goal of the multithread version is to divide the workload of the denoising filter, decreasing the time required to process a complete point cloud frame. As Figure 4.4 depicts, the filter's workload splits across N threads, where the default value of N is four because of the number of Central Process Unit (CPU) cores present in the used platform.

$$StartPoint = \frac{PointCloudSize}{NumberOfThreads} * ThreadNumber \tag{4.1}$$

$$EndPoint = \frac{PointCloudSize}{NumberOfThreads} * (ThreadNumber + 1) \tag{4.2}$$

Each thread filters a point cluster block defined by Equation 4.1, the start point, and Equation 4.2, which defines the processing limit. Moreover, a *filter point* method is used to filter the point cluster, which, depending on the configuration, can be DROR, LIOR, or DIOR filters. Lastly, when the worker threads

complete their execution, the filtered point cloud is reconstructed and transmitted to the next module, the *ROS point cloud encoder*.

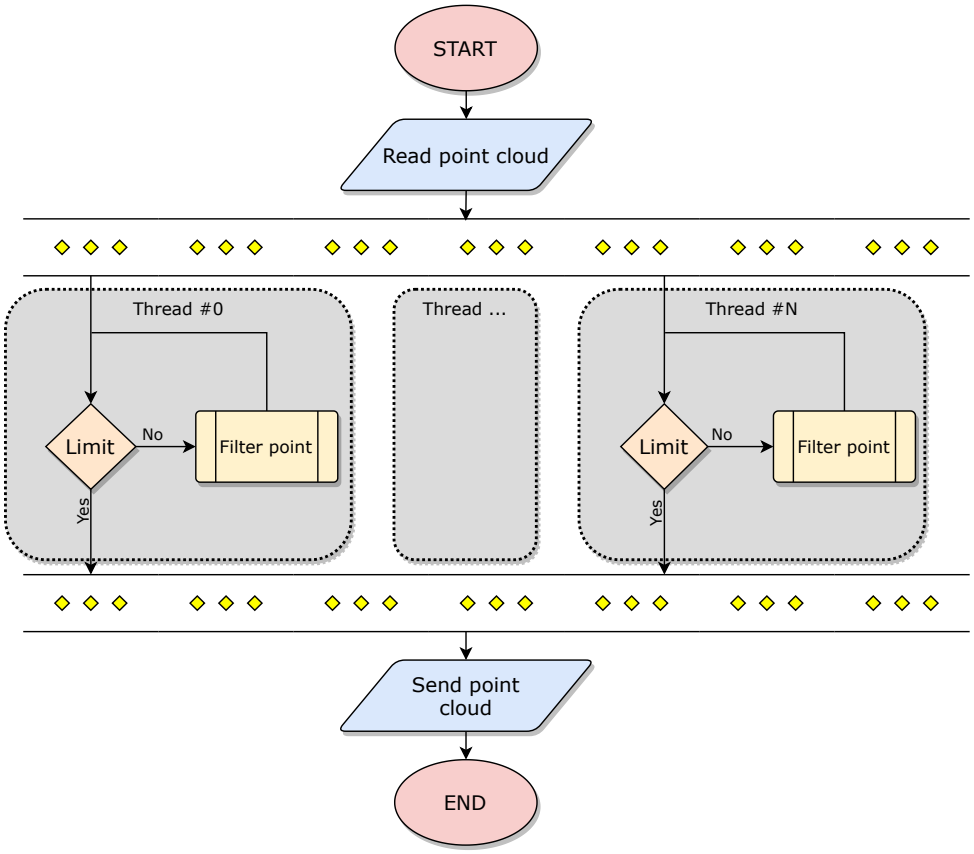


Figure 4.4: ALFA-Pd multithreading configuration.

Filters Implementation

The workflow of the software modules depends on the user configuration. If the user selects ROR, SOR, or VG as the denoising filter, their direct PCL implementation is used. In contrast, if DROR, LIOR, or DIOR are selected, their implementation is employed to filter each point in the point cloud.

As depicted in Algorithm 1, DROR removes points based on their distance to the LiDAR sensor. The DROR filter can be tuned using specified parameters, which are the point cluster (P), a minimum search radius (SR_{min}), and a minimum of neighbors (n_{min}). The point cluster is a set of points allocated to the thread running the filter point method in each iteration, and the SR_{min} prevents points from being removed when they are very close to the sensor. Finally, the n_{min} parameter defines the minimum of close neighbors that a point requires to be an inlier.

To filter a point cloud, DROR must go through all point cloud points. The distance of each point to the sensor is calculated using Equation 4.3. Subsequently, if the distance to the sensor is below the search radius threshold SR_{min}, $R1$ takes the value of SR_{min}. Conversely, $R1$ is set using Equation 4.4. Henceforward, the number of neighboring points is determined for each point using the PCL module

Algorithm 1 DROR filter point pseudocode

Require:

P : Point cloud cluster
n_{min} : Minimum number of neighbors
SR_{min} : Minimum search radius.
$NeighborSearch$: PCL module that calculates the nearest neighbors

1: **for** $p \in P$ **do**
2: **if** $r_p < SR_{min}$ **then**
3: **R1** $\leftarrow SR_{min}$
4: **else**
5: **R1** $\leftarrow SR_p$
6: **end if**
7: $n \leftarrow NeighborSearch(p, \textbf{R1})$
8: **if** $n > n_{min}$ **then**
9: $Inliers \leftarrow p$
10: **else**
11: $Outliers \leftarrow p$
12: **end if**
13: **end for**

NeighborSearch, which determines if a point is an inlier or an outlier using the calculated $R1$. If the number of neighbors exceeds the n_{min} threshold, the point is categorized as an inlier; otherwise, it is classified as an outlier.

$$SR_p = \beta \times (r_p \times \alpha) \tag{4.3}$$

$$r_p = \sqrt{x_p^2 + y_p^2} \tag{4.4}$$

The LIOR filter was implemented based on the information provided by Park et al. [62]. LIOR premise is that noisy outliers have less reflected intensity than neighboring inlier points. Therefore, LIOR can classify a point as an inlier or outlier based on a simple comparison and threshold verification. As shown in Algorithm 2, the LIOR filter requires fewer conditions than others to operate on a point cloud frame. It requires the point cluster P, the point cloud block corresponding to the thread on which it is executing, the minimum of neighboring points (n_{min}) required to categorize a point as an inlier or outlier, and the intensity threshold (I_{thr}), used in point classification. This threshold is defined based on the used sensor specifications provided by the manufacturer due to different standards between them. Furthermore, LIOR uses a fixed search radius (SR), requiring it to be defined before execution. Similar to DROR, the software implementation of this algorithm uses the NeighborSearch module of the PCL library.

After the first pre-evaluation based on the point's intensity, LIOR reevaluates the point cloud outliers with a second step, where a ROR type filter is applied. This step checks the number of points inside a

Algorithm 2 LIOR filter point pseudocode

Require:
P : Point cloud cluster
n_{min} : Minimum number of neighbors
I_{thr} : Minimum generic intensity threshold
SR : Search radius.
$NeighborSearch$: PCL module that calculates the nearest neighbors

```
1: for p ∈ P do
2:     if I_p > I_thr then
3:         Inliers ← p
4:     else
5:         n ← NeighborSearch(p, SR)
6:         if n > n_min then
7:             Inliers ← p
8:         else
9:             Outliers ← p
10:        end if
11:    end if
12: end for
```

fixed radius. If the number of points inside the radius exceeds the threshold, the point is considered an inlier; otherwise, the point is considered an outlier.

Being a combination of DROR and LIOR, DIOR uses the same execution requirements as its predecessors. It requires a point cluster P, the minimum of neighbors n_{min} to validate points, a generic intensity threshold I_{thr}, and a minimum search radius SR_{min}. DIOR also uses the same PCL modules as DROR and LIOR. As depicted in Algorithm 3, the *filter point* method removes the outliers from a given set of points. First, the system verifies the intensity value of a point. If (I_p) is above the intensity threshold, the system considers it an inlier, discarding further processing. However, if the point fails in the intensity verification, a search radius ($R1$) is calculated, similarly to DROR. Finally, a neighbor search is performed using a PCL method to determine whether the point has enough neighbors (n) to be verified as an inlier.

Algorithm 3 DIOR filter point pseudocode

Require:

P : Original point cluster

n_{min} : Minimum number of neighbors

I_{thr} : Minimum generic intensity threshold

SR_{min} : Minimum search radius.

$NeighborSearch$: PCL module that calculates the nearest neighbors

```
for p ∈ P do
    if I_p > I_thr then
        Inliers ← p
    else
        if r_p < SR_min then
            R1 ← SR_min
        else
            R1 ← SR_p
        end if
        n ← NeighborSearch(p, R1)
        if n > n_min then
            Inliers ← p
        else
            Outliers ← p
        end if
    end if
end for
```

4.2 ALFA-Pd Memory

ALFA-Pd memory module interfaces the software modules running on the PS and the hardware modules running on the PL. Moreover, ALFA-Pd memory is responsible for storing all the information accessed both from the PL and the PS. This information includes point cloud data, configuration data, and signaling. There are two different types of memory present in the platform. The BRAM memory module provides faster execution times, while the DDR module features a more generic implementation, offering more portability.

4.2.1 BRAM Implementation

Data transmission between the software and the hardware layers represents one of the largest bottlenecks of software/hardware co-design systems. Furthermore, denoising algorithms need to perform multiple accesses to the same data point, which worsen this problem. To address this issue, the BRAM version utilizes specific memory regions in the hardware to store entire point clouds, allowing faster execution times in exchange for less portability.

The BRAM-based implementation is optimized for performance since the system can read and output data within one clock cycle. As described in Figure 4.5, the memory layout features four independent sections, each correspondent to one BRAM, and containing information about an individual point's parameter. The first position of each section is reserved for configuration purposes, which are used by the hardware accelerators. The *X BRAM*, alongside the x coordinates for all points, holds the point cloud size, which is necessary for the hardware modules to know the number of points it needs to process. Conversely, the first position of the *Y BRAM* stores the start signal, used by the hardware layer to signal a new frame to process, while in the *Z BRAM*, the same position signals when the hardware accelerators finish the filtering tasks. Finally, the *I BRAM* stores the filter selector. ALFA-Pd supports the hardware-accelerated versions of DROR, DIOR, and LIOR, being possible to change the filter running upon execution time using the value stored in the filter selector.

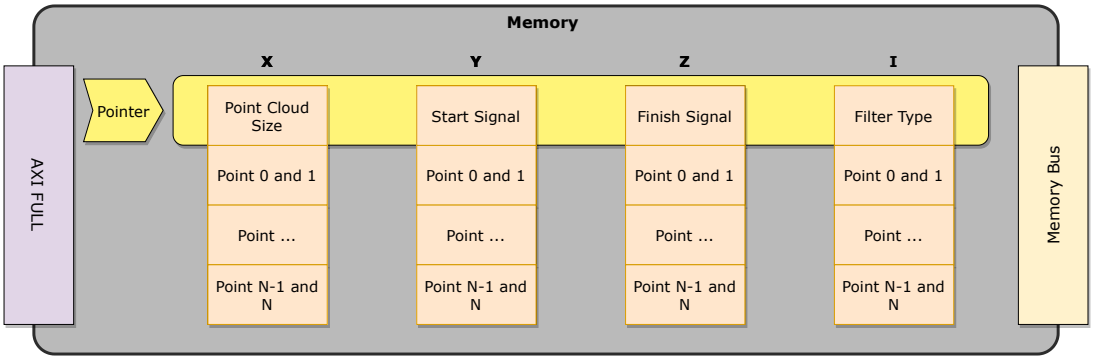


Figure 4.5: ALFA-Pd BRAM memory architecture.

The ALFA-Pd BRAM memory module has a shared pointer pointing to all four sections, maintaining synchronism between them. Each position inside these sections contains 32-bits of information. Therefore, by having a shared pointer that accesses the 4 BRAM modules simultaneously, it is possible to obtain 128-bits worth of data per operation, which equals to two data points. As depicted in Figure 4.6, the first 16-bits of each BRAM module store the information on the respective parameter of a point, and the following 16-bits contain the same parameter belonging to the next point. Furthermore, every point stored in the less significant bits has a pair index; for example, in memory position one, the first 16-bits of memory holds the information of point zero, and the most significant bits hold information of point one. The ALFA-Pd hardware uses this information to determine the precise location of a particular point in the memory region.

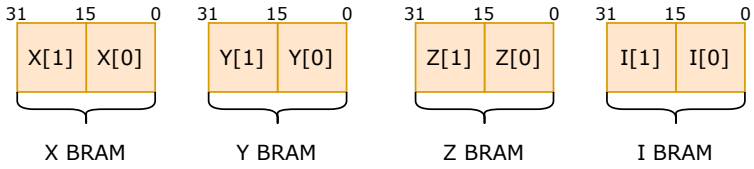


Figure 4.6: ALFA-Pd BRAM point layout.

Figure 4.7 depicts a block diagram of the whole BRAM implementation system. The memory modules (2) are referent to the mentioned memory sections, which are implemented using 4 block memory generators [70], one for each section. These block memory generators are a Xilinx-provided advanced memory constructor Intellectual Property (IP), which generates performance-optimized memories utilizing integrated BRAMs found in Xilinx FPGAs. This IP core contains two separate ports, port A and port B, allowing a true dual-port configuration. The dual-port configuration enables read and write operations simultaneously. Furthermore, the dual-port configuration is connected to multiple modules to take advantage of this feature. One port connects to the PS through AXI-Full, enabling the ALFA-Pd embedded software modules to write and read the BRAM. The other port links the BRAMs to the memory interface by using a custom memory bus. This setup enables the PS and the PL modules to exchange data. The dual-port functionality also simplifies signal processing because both the PS and the PL have read and write permissions, allowing the PS and the PL to clear the end and start signal.

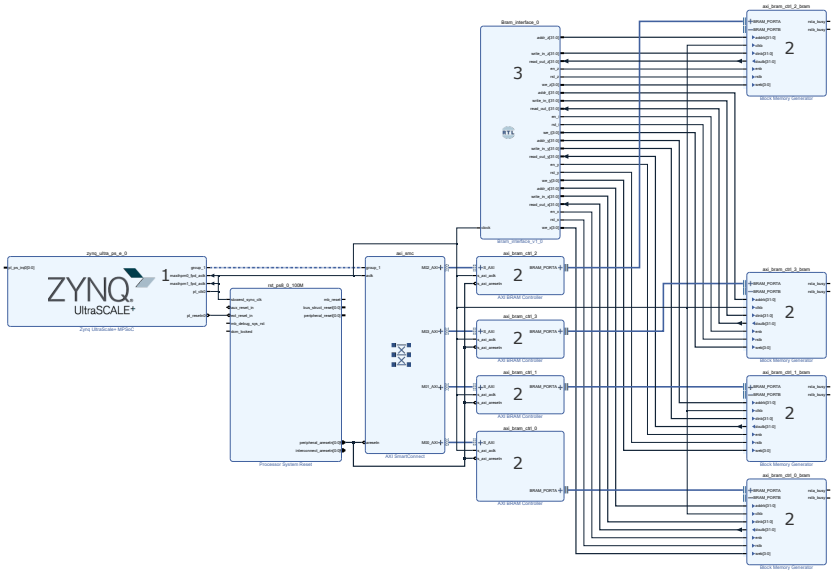


Figure 4.7: ALFA-Pd BRAM implementation block diagram.

The hardware weather denoising modules (3), are encased inside the memory interface module (further explained), which is responsible for communicating with all external modules. Additionally, the memory interface is responsible for controlling all block memory generators' B ports, where each has its dedicated bus. Despite the BRAM bus used by default being 32-bits, it is possible to use buses with more width, enhancing compatibility with different block memory generators' configurations. Using a 32-bits

bus implies that the memory interface module fetches 2 points per clock cycle from the BRAMs. Nevertheless, in other platforms that have different architectures, it is possible to increase the BRAM bus size, which increases the number of points obtained per clock cycle. For example, using a 64-bit wide bus, the system can fetch 4 points per clock cycle, enhancing the performance even further.

BRAM Memory Interface

The memory interface module is responsible for interfacing with the BRAM memory modules and the hardware controller. The memory interface uses a custom bus to communicate with the BRAM modules, which allows the transaction between the modules to be complete within one clock cycle, simplifying the control mechanism. The state machine, represented in Figure 4.8, features five states and is responsible for controlling the BRAM memory interface. The module begins execution in the *Stopped State*, awaiting configuration and the start signal. When the PS provides the start signal, the state machine progresses to the *Update cache* state, which is responsible for updating this module's caches.

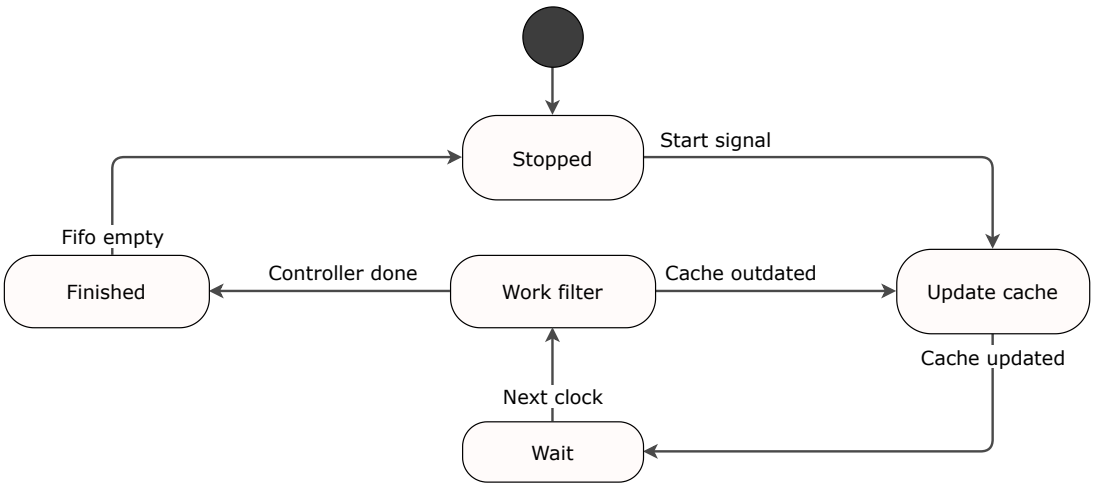


Figure 4.8: ALFA-Pd BRAM memory interface state machine.

There are two caches in the memory interface: a Point Cluster (PC) cache and a Neighbor Finder (NF) cache. The size of the PC cache is proportional to the number of PCs, whereas the size of the NF cache is proportional to the number of NFs. The hardware controller uses the PC cache to reload the PC when the processing phase is completed and the point stored by the PC is obsolete. The NF modules utilize the NF cache to compute the distance between the point in the cache and the point stored in the PC, and to classify the points as inliers or outliers.

The BRAM modules store two points per memory address. As a result, the number of clock cycles necessary to fill the caches is always half of the cache size. As the cache increases, the fetching procedure takes longer, reducing performance. However, increasing the cache size allows the deployment of more PC/NF modules, which heavily parallelizes execution, mitigating the time consumed by the fetching procedure.

After receiving the start signal from the PS, the status is changed to *Update cache*, which initiates the ALFA-Pd workflow. Subsequently, after updating the cache through point fetches from the BRAM modules, the execution state is set to *Wait*. In the *Wait* state, the module performs a clock cycle delay to guarantee that subsequent modules remain in sync. After this, the state changes to the *Work filter* state. The *Work filter* state begins de-caching the points while simultaneously updating the NF cache, boosting overall throughput. The mentioned operations repeat until the hardware controller completes filtering the point cloud.

When the hardware controller finishes its tasks, the executing state changes to *Finished*. If the hardware module detects an outlier, it keeps its index in a First In First Out (FIFO), which is then utilized by the interface to eliminate all the detected outliers. Therefore, in the final state of the state machine, the memory interface reads the outlier indexes (memory positions) from the FIFO and discards them. When the FIFO is empty, the hardware module triggers the PS that the processing has finished, at which point the module returns to the *Stopped* state, waiting for the next frame to process.

4.2.2 DDR Implementation

The DDR version seeks to provide a more general implementation by utilizing a more generic memory design. Rather than using BRAMs to fetch/store the point cloud, this version uses DDR memory. It utilizes a AXI-full interface to acquire all the point cloud data, whereas the AXI-lite interface is used for transmitting signals and configurations associated with the processing of a point cloud frame.

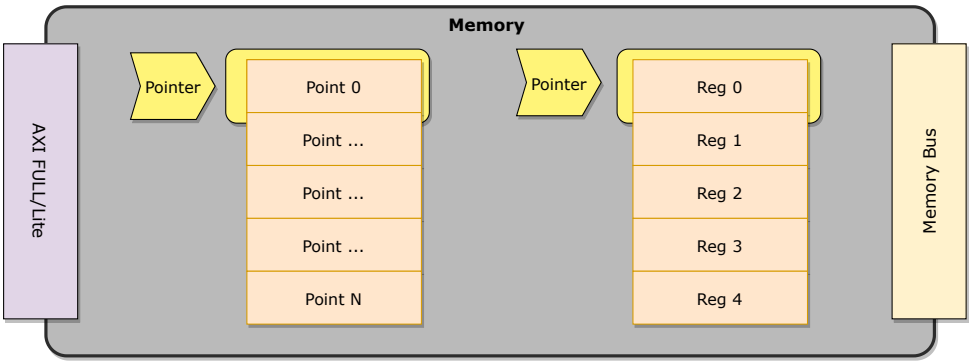


Figure 4.9: ALFA-Pd DDR memory architecture.

Using the DDR memory eliminates the need for a dedicated memory to store the points since the PL can use the PS memory domain. However, the time required by the hardware accelerators to access the memory is higher because the DDR is closer to the CPU than the PL fabric. Unlike the BRAM implementation, the DDR version does not use several memory sections to separate the points' parameters, using instead only one section, as depicted in the left side of Figure 4.9. Each memory position holds a point with 16-bits width parameters, which results in each position having 64-bits. Additionally, an AXI-lite module with four registers is utilized to signal and configure the ALFA-Pd hardware module (right side).

This module communicates with the remaining of the ALFA-Pd hardware modules through a custom bus, allowing access to specific registers.

The used AXI-full interface operates on a 64-bit bus, allowing it to fetch a point per transaction. As illustrated in Figure 4.10, the point composition progresses from the less significant 16-bits, that define the point's X parameter, to the most significant 16-bits, which represent the point's intensity value. Additionally, by representing the coordinates with a 16-bits representation, the maximum distance value that a parameter can hold is 327 meters.

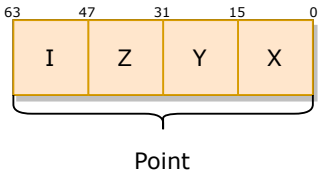


Figure 4.10: Point memory placement.

The hardware features an AXI-lite interface, which allows the PS to configure and control the hardware modules easily. The AXI-lite module contains four registers that include all the data necessary for processing a point cloud frame. Each register has 32-bits, so the PS concatenates all the configurations, allowing for a more efficient use of the memory space. The parameters required for processing a point cloud frame are depicted in Figure 4.11. Firstly, register 0 stores all the necessary configuration and control signals. The hardware can only read from this register, moving complete hardware control to the PS or other module connected to the AXI-lite interface. Register 1 stores the point cloud size of the executing frame. The ALFA-Pd's hardware modules use the point cloud size to check when the filtering process finishes. Additionally, register 2 carries the parameter *Finish info*, the number of outliers produced by the hardware modules after the filtering. Finally, to maintain synchronism between the PL and the PS registers 3 and 4 store the current *frame Id* of each module. When the PS *frame Id* differs from the PL *frame Id*, the PL starts the processing tasks since a new frame is available. Moreover, when the PS and the PL *frame Id* are the same, the PS recognizes that it can transmit a new frame to the PL for filtering.

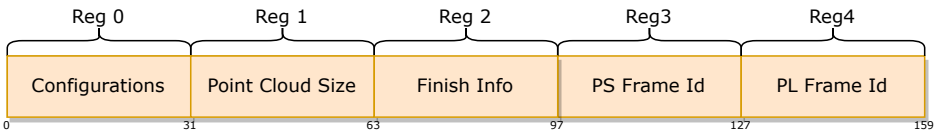


Figure 4.11: AXI-Lite registers.

In Figure 4.12, the register 0's layout can be observed. The first bit is allocated to the *start signal*, and the next 4 bits represent the *filter selector*, a variable responsible for changing what filter is executing. The section from bit 6 to bit 9 holds the *intensity threshold* used by DIOR and LIOR algorithms. The following 4 bits can have different meanings depending on the filter used. It acts as a fixed radius for the LIOR filter's neighbor search and as the minimal search radius for the DROR or DIOR algorithms. The parameter *min*

neighbor threshold is encoded in bits 13 to 21. Finally, the remaining bits hold the multiplication parameter, which allows calculating the compensation for the point spacing increase resultant from surfaces that are not perpendicular to the LiDAR beams.

Figure 4.12: AXI-Lite configurations register.

This parameter translates to β in Equation 4.3. The constants β and α are usually below 1, meaning that they are decimal variables, which presents challenges in the hardware. Therefore, instead of using Equation 4.3 as the software version, the simplified version in Equation 4.5 is used in the hardware.

$$SR_p = \beta \div (r_p \div \alpha) \tag{4.5}$$

Regarding the structure of the DDR version, it follows the same approach as the BRAM implementation into: software, memory, and hardware, which are depicted in Figure 4.13. The memory modules (2), are made up of two components: *memoryInterface_AXI* and *AXI_lite_slave*. Firstly, the *memoryInterface_AXI* manages the AXI-Full communication, enabling read and write operations within the DDR memory area. Additionally, this module has a read burst capacity of 32 operations, fetching 32 points in each transaction. On the other hand, the *AXI_lite_slave* module is responsible of handling all memory interface signals and configurations. The hardware weather denoising modules (3), encapsulate all the hardware denoising accelerated methods. They receive the starting signal from the AXI Lite module and send back the result through the same module. They obtain point cloud data through the *memoryInterface_AXI*, which is also responsible for removing noise points from the DDR memory.

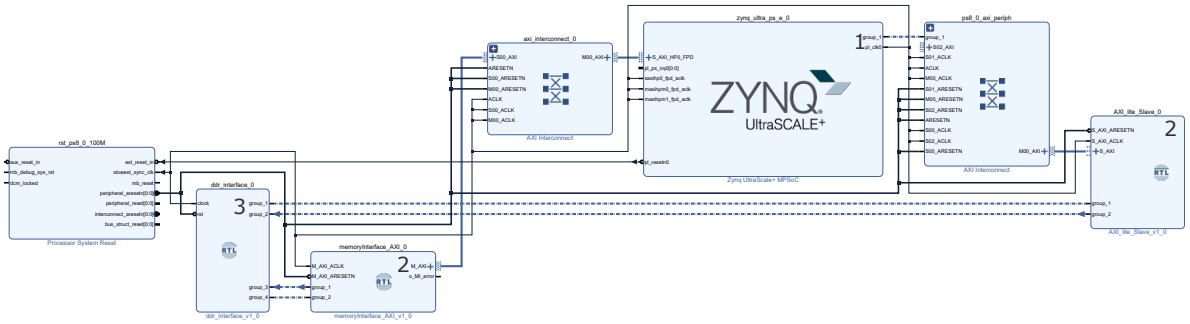


Figure 4.13: ALFA-Pd DDR implementation block diagram.

Table 4.2 depicts the DDR memory interface module's parameters. The interface allows changes in the burst size to enable compatibility for different configurations of the AXI module. However, the number of PC must always be equal to or smaller than the burst size since the points obtained in the burst transaction will fill a cache that contains the reloading points to each PC. Executing parallelly to the AXI burst transactions, the NF modules use points to find neighboring points. Therefore, because the number

of cycles needed to do an AXI burst transaction is always greater than the number of points obtained in the transaction, the number of NF modules is set to one.

Table 4.2: Default parameters of DDR version.

Paremeter	Default value
AXI Burst size	32
AXI cluster cache multiplier	1
Point Clusters number	16
Neighbor Finders number	1
DDR Base address	0x0F000000

DDR Memory Interface

The Memory Interface module is responsible for interfacing the DDR memory, where the points are stored, and the hardware controller, which controls the execution flow of all the connected hardware modules. Figure 4.14 depicts the state machine used by the DDR memory interface. The module begins execution in the *Stopped* state, and waits for configurations and a start signal. When the PS provides the start signal and its *frame Id* differs from the PL, the interface starts the execution by shifting to the second state, *fetch AXI cluster cache*. In this state, the hardware controller module updates one of the DDR memory's cache. The DDR memory interface employs a two-cache architecture, one for the point clusters and one to feed the neighbor finder.

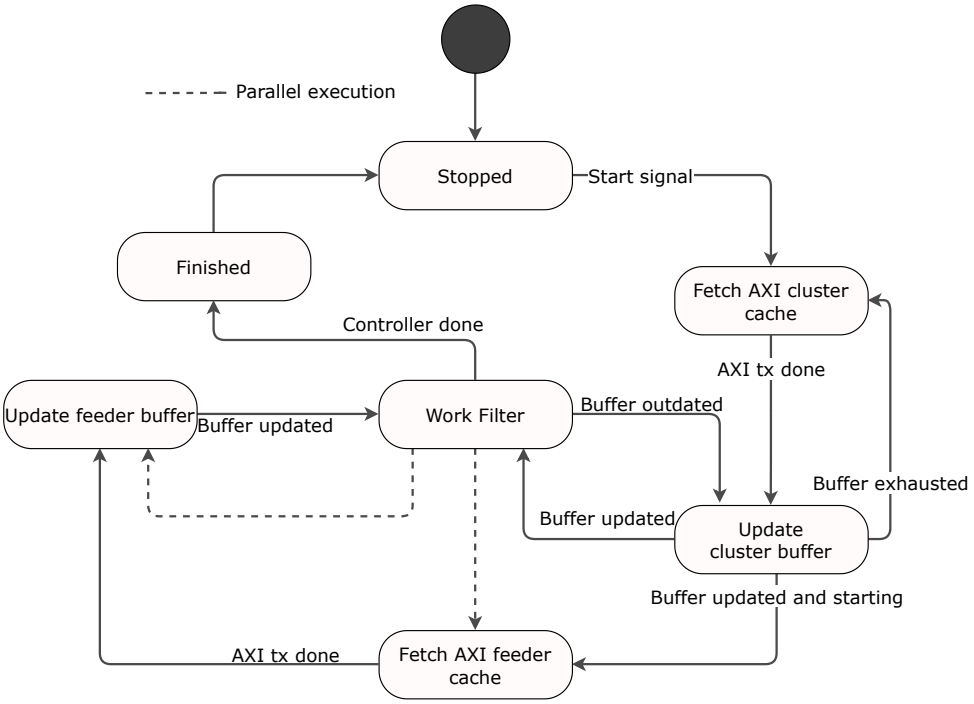


Figure 4.14: ALFA-Pd DDR memory interface state machine.

The DDR memory interface employs a cache denoted by AXI cache in Figure 4.15. This cache purpose is to hold values obtained through an AXI transaction. Since an AXI transaction is configured to read a point of 32-bits from the DDR, the AXI cache must be at least as long as the AXI transaction used to get the points, in factors of two. For example, if the transaction size is 32 points and the number of point clusters is 42, the cache must hold 64 points.

The number of PC in use defines the cluster buffer size. However, as depicted in Figure 4.15, this buffer size must always be smaller than the AXI cache since it acts like a sliding window of the AXI cache. The feeder buffer design also works as a sliding window to the AXI feeder cache, holding points to reload the neighbor finder modules. Moving the AXI caches' points to the smaller buffers takes just one clock cycle, which speeds up execution until the caches are emptied, meaning that there are not enough points to move to the cluster buffer or the feeder buffer.

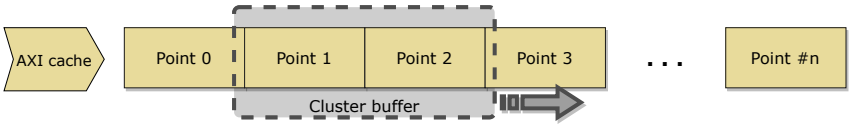


Figure 4.15: ALFA-Pd memory interface cache design.

The caches must be updated after they depletion, which results in a change of the execution state. Sequentially, when the cluster buffer is updated, the state machine can go to two different states, depending on the condition. If the feeder buffer is still up to date, the state machine progresses to the *Work filter* state. Otherwise, the next executing state must be the *Fetch AXI feeder cache*, since the AXI feeder cache is empty. The *Fetch AXI feeder cache* works like the *Fetch AXI cluster cache*, but instead of saving the data received through the AXI transaction into the AXI cluster cache, it stores the values in the AXI feeder cache. When the AXI transaction ends, the state machine executes the *update feeder buffer* state. The *Update feeder buffer* state execution flow is identical to the *Update cluster buffer* state execution flow, except that it uses a sliding window to go through all of the AXI feeder cache's points. After the update operation is complete, the state machine moves to the *Work filter* state.

The *Work filter* state is responsible for controlling the execution flow of the hardware controller module, which is does the denoising of the point cloud. The hardware controller features a stop and resume mechanism to guarantee that it remains synced with the memory interface, which are manipulated in the *Work filter* state. Additionally, when in this state, the module parallelizes some tasks, increasing the overall throughput of validated points. As depicted in Figure 4.16, while the hardware controller executes a process cycle, the memory fetches the AXI feeder cache and reloads the feeder buffer.

When the hardware controller finishes, it signals the memory interface, and the executing state changes to the *Finished* state. In this last state, every outlier stored in the FIFO of the hardware module is removed from the DDR memory. Moreover, the points are removed using the AXI-full module, which sets the points indexes, obtained from the FIFO, to zero. When the FIFO is empty, the executing state changes to the *Stopped* state, waiting for a new frame to process.

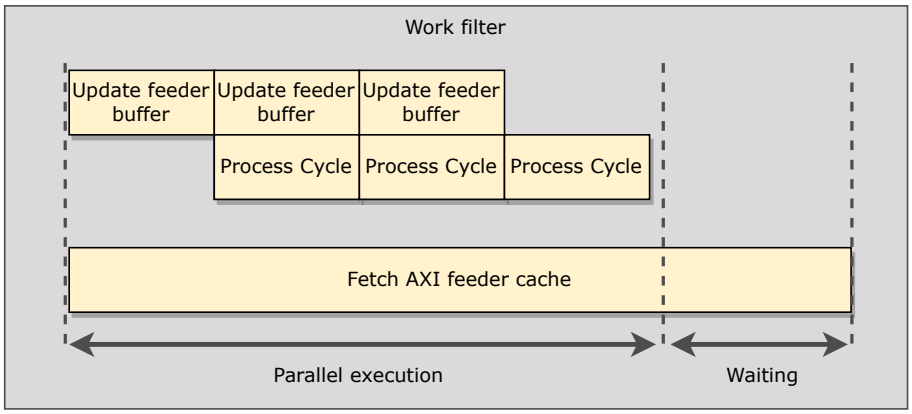


Figure 4.16: ALFA-Pd memory interface parallel execution.

4.3 ALFA-Pd Hardware

One of the key points of the ALFA-Pd framework is the ability to deploy customized hardware accelerators on FPGA fabric, openning up the possibility to enhance the performance of the supported software versions of weather denoising algorithms. ALFA-Pd's hardware implementation, as seen in Figure 4.17, consists of four major components: (1) the hardware controller; (2) a memory interface; (3) several PC blocks; and (4) NF units. Each PC block requires at least one NFs, each of which requires one Distance Calculator module from the ALFA core accelerators.

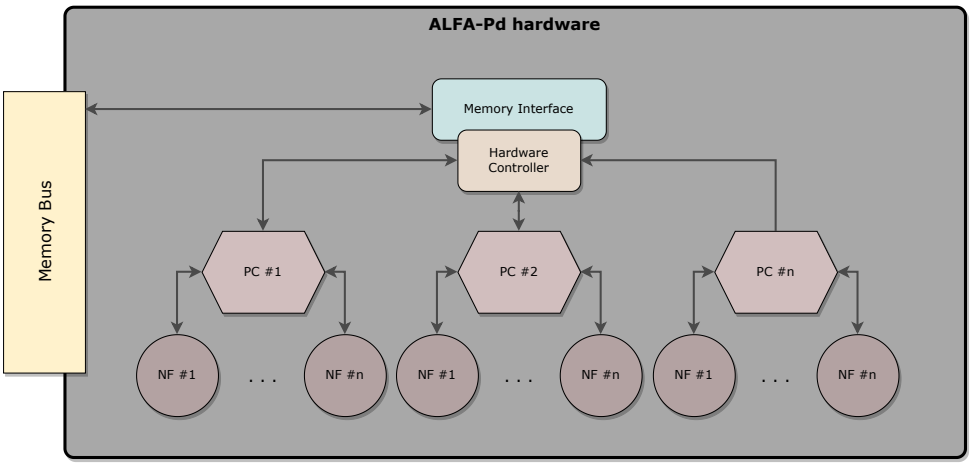


Figure 4.17: ALFA-Pd hardware modules architecture.

The PC and NF blocks parallelize the algorithm's execution in finding neighbors, which classify a given point as inlier or outlier, decreasing the required time to process a point cloud frame. The number of PC and NF dictates the filter performance at a resource cost. Moreover, the platform chosen dictates the limit of modules that can be deployed. Thus, in an ideal world, with unlimited resources, the number of PC and NF would be the size of the point cloud to process the whole point cloud in one clock cycle.

Hardware Controller

The Hardware Controller is the main block of the denoising accelerator. This module is responsible for: sending points to the PCs for comparison, checking and controlling the output of all PCs, and tagging/storing the points classified as outliers from each PC. The hardware controller does not change depending on the memory implementation and acts as an abstraction layer for the upper modules. Additionally, it has a start/stop mechanism to maintain synchronism with the caches controlled by top modules.

The hardware controller is in charge of performing the appropriate operations depending on the PC's results. It has a built-in FIFO that stores the point index of each outlier until the memory interface module removes it. As seen in Figure 4.18, the hardware controller module evaluates each PC output in each clock cycle. Since the module can only write one point in the FIFO per clock cycle, it buffers point indexes to subsequently transferred them to the FIFO. The FIFO buffer ensures that no point is lost if more than one is classified as an outlier in the same cycle. If the buffer contains an outlier, the hardware module saves it in the FIFO independently, regardless the operation state.

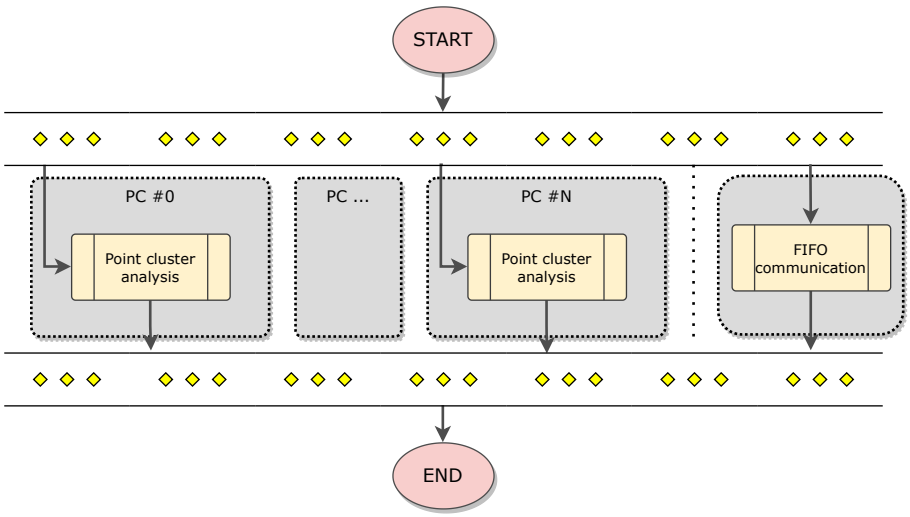


Figure 4.18: ALFA-Pd hardware controller execution flow.

When the PC ends validating a point, all the information, such as the point itself and the indexes becomes outdated. The process of reloading the PC information is presented in Algorithm 4. First, the hardware controller sends the new point to process into the PC. Using cluster buffer from the memory interface, the point selected to reload a specific PC is determined by the number of PCs that finished executing before it. Therefore, after processing the point, the *finish_counter* is also updated. Note that all point cluster analyzers are executed simultaneously and synchronously.

Algorithm 4 Point cluster output analyzes pseudocode

```
1:  if inlier or outlier then
2:      if outlier then
3:          fifo_buffer[fifo_size] ← cluster_point_index
4:          fifo_size ← fifo_size + 1
5:      end if
6:      cluster_point_index ← point_index
7:      point_index ← point_index + 1
8:      cluster_point ← cluster_buffer[finished_counter]
9:      finished_counter ← finished_counter + 1
10: end if
```

Point Cluster

The PC module is responsible for evaluating the point under processing, i.e., evaluate if it is an outlier or not. If the number of neighbors identified is less than the defined neighbor threshold, the point under validation is classed as an outlier; if the number of neighbors found is more than the given threshold, the point is categorized as an inlier. As depicted in Figure 4.19, three submodules divide the PC module.

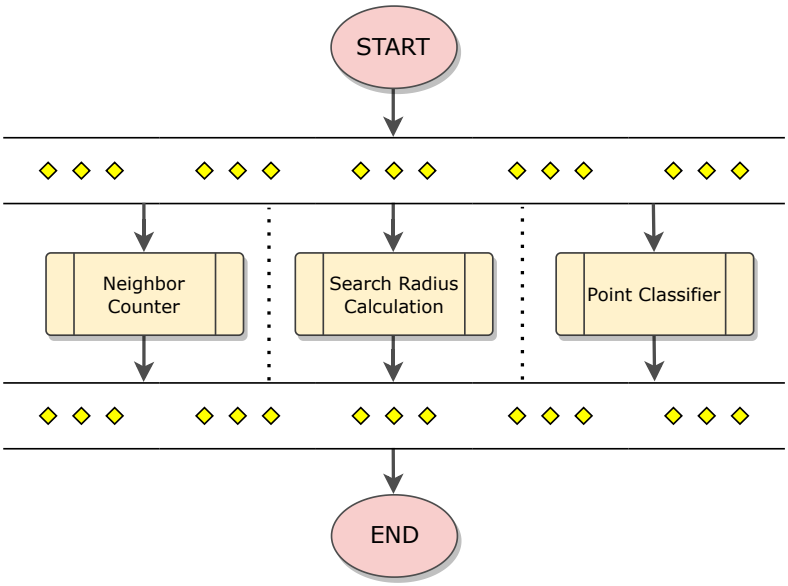


Figure 4.19: ALFA-Pd point cluster execution flow.

The first module is the neighbor counter, a simple module that counts the number of neighboring points a PC has. As Algorithm 5 depicts, the neighbor counting module compares the distance calculated by each NF with the search radius to determine if it is smaller than it. If in fact is smaller, then a neighbor, and the *neighbor counter* is incremented; otherwise, the computed distance is discarded.

The search radius calculation module determines the radius $R1$ based on the filter specified. F_s represents the filter selected. If it contains the number 1, the PC uses a fixed search radius, used by

Algorithm 5 Neighbor counter pseudocode

```
for neighbor_index = 1, 2, 3...N do
    if distance[neighbor_index] <= R1 then
        neighbor_counter ← neighbor_counter + 1
    end if
end for
```

LIOR. Otherwise, the PC uses a dynamic search radius, used by DROR and DIOR, in which the Algorithm 6 depicts the needed calculation. If the point distance to the sensor r_p is below the predefined threshold SR_{min}, then the value of the search radius R1 is set to the SR_{min} , otherwise, R1 assumes the value of the SR_p.

Algorithm 6 Search radius calculation pseudocode

```
if F_s != 1 then
    if r_p < SR_min then
        R1 ← SR_min
    else
        R1 ← SR_p
    end if
else
    R1 ← SR_LIOR
end if
```

The point classifier submodule is responsible for tagging points as inlier and outliers. As shown in Algorithm 7, the submodule first determines if the filter is either DIOR or LIOR, which needs the point's intensity. If this intensity is greater than the threshold I_{thr}, the point is categorized as an inlier. The classification of inliers when the point reflectivity exceeds the intensity threshold takes one clock cycle to complete, making it the system's quickest validation scenario. Therefore, if every point is validated as an inlier through the intensity condition, the number of cycles required to do the point cloud validation is the point cloud size, not accounting for the time lost in AXI transactions.

If a point does not pass the intensity validation, either because of the threshold verification or because the employed filter does not use it, the point must have more neighbors than the $neighbor\_threshold$ to be classed as an inlier. However, if the point lacks the required neighbors, it is necessary to determine if it is an outlier. If the number of points compared p_{comp} is greater than the size of the point cloud P_{size}, it means that the point was already compared with the entire point cloud, indicating that it is an outlier. The number of NF modules determines the number of comparisons performed every clock cycle. Thus, increasing the number of NF modules severely boosts the system's performance.

Algorithm 7 Point classifier pseudocode

```
1:  if F_s >= 1 & I_p > I_thr then
2:      inlier ← 1
3:      outlier ← 0
4:  else
5:      if neighbor_found >= neighbor_threshold then
6:          inlier ← 1
7:          outlier ← 0
8:      else
9:          inlier ← 0
10:         if p_comp > P_size then
11:             outlier ← 1
12:         else
13:             outlier ← 0
14:         end if
15:     end if
16:     p_comp ← p_comp + NF
17: end if
```

Neighbor Finder

The PC uses the NF module to calculate the distance between two given points, using its output to classify the points considering the neighbor threshold and the search radius R1. As noted, each PC can have multiple NF modules, each totally independent from the others. The NF module uses the feeder cache to compute the current point with the stored ones. As seen in Figure 4.20, different NFs do comparisons using the same point. Thus, the feeder cache supplies points to all NFs each cycle, ensuring that the point cloud is continually moving ahead. The feeder cache is independent from the NF and PC modules since it works as a circular buffer for the point cloud, ensuring that all points flow through the NF module.

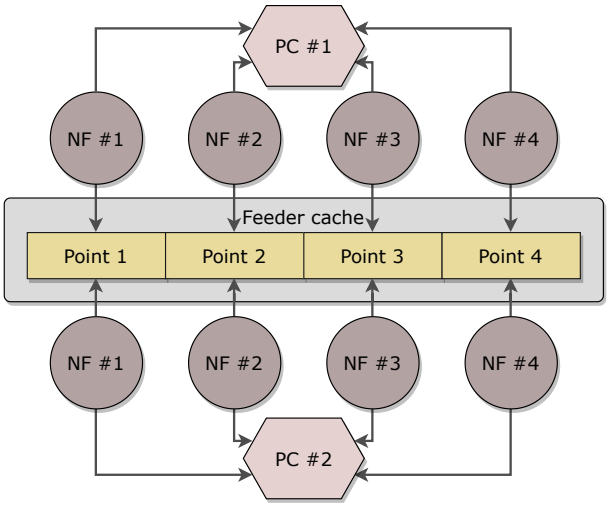


Figure 4.20: Neighbor finder communication.

The NF module, which is the core calculation module of the ALFA-Pd hardware accelerator, calculates the distance between two points in a single clock cycle. Algorithm 8 illustrates the method for calculating the distance between two points: the point under evaluation, stored inside the PC, and a point provided by the feeder cache. Firstly, the NF module computes the three-dimensional vector of the two points. To successfully calculate the vector's square value, the point must be positive, otherwise, the calculation would result in an overflow. To address this issue, the NF module examines the value of the most significant bit, which indicates whether or not an integer is negative, and calculates its absolute value based on the result.

Algorithm 8 Distance calculation pseudocode

1: $Vector \leftarrow p_1 - p_2$
2: **if** $Vector[15] == 1$ **then**
3: $\quad Vector = -Vector$
4: **end if**
5: $distance \leftarrow P_{dist}$

4.4 ALFA-DVC

The ALFA-Debugger Visualiser Configurator (DVC) tool, depicted in Figure 4.21, is a high-level QT application that runs on a desktop system. It supports real-time visualization of up to two point clouds simultaneously, the deployment and evaluation of software-based algorithms, and debugging/configuration of those weather denoising methods.

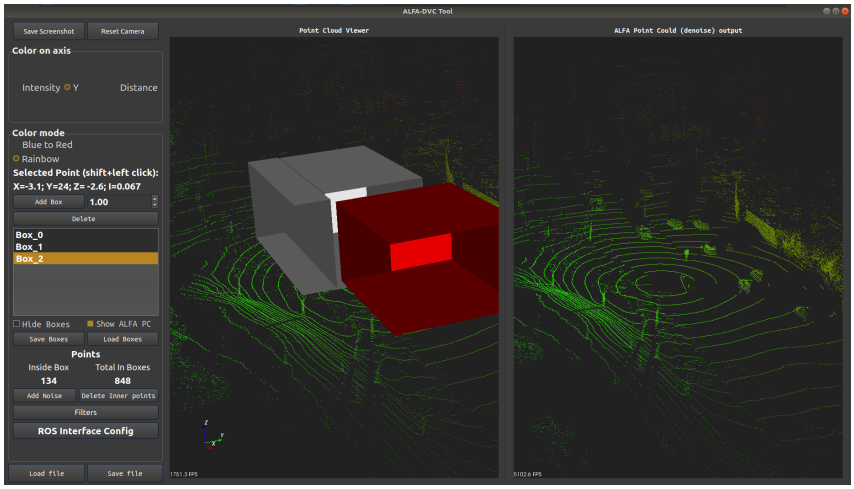
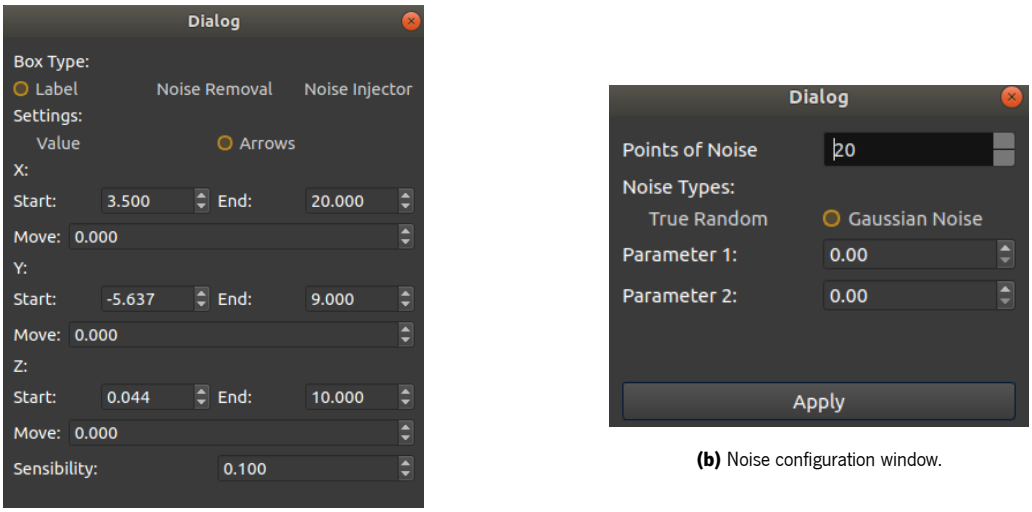


Figure 4.21: ALFA-DVC user interface.

Point cloud data can be loaded into the ALFA-DVC tool using Point Cloud Data (.pcd) and Polygon File Format (.ply) files stored in the file system or through the subscription of ROS topics. These topics must

use the PointCloud2 message type, which is the standard for point cloud data transmission inside ROS. The application allows storing screenshots of the point cloud, as well as publication of new point clouds to a ROS topic. Additionally, the point parameters, such as coordinates and intensity values, are also accessible in the interface by pressing shift + left mouse button, allowing quick and simple performance evaluation of weather denoising algorithms.

ALFA-DVC enables easy interaction and analysis of point clouds via a box system, in which the user can create/delete and position boxes inside the point cloud, as illustrated in Figure 4.21. Moreover, it is possible to save/load box configurations from the file system, which allows for easier repeatability between evaluation sessions.



(a) Box configuration window.

(b) Noise configuration window.

Figure 4.22: ALFA-DVC subwindows.

Three types of boxes are available: (i) label boxes, (ii) noise removal boxes, and (iii) noise injection boxes. By double-clicking on the desired box, it is possible to alter its behavior, coordinates, and size, as Figure 4.22a illustrates. While the label boxes count the number of points within their area and label them as noise points to evaluate algorithms performance, then the noise removal boxes remove all the points inside them, cleaning a point cloud of weather noise. The noise injection boxes create a random number of noise points, which are then injected into the point cloud to simulate noise. When data points are collected under ideal weather circumstances, ALFA-DVC can simulate and generate different types of weather noise for specific portions of the point cloud using the box system. This can be accomplished by simply clicking on the corresponding button, which opens a small window, illustrated in Figure 4.22b, displaying all the setting options for simulating point cloud noise.

Finally, the filter window enables the configuration, application, and assessment of denoising methods in a streamlined manner. This window, displayed in Figure 4.23, is where all the interaction with everything related to the weather denoising filters is done. The user can choose which filter to apply from a list of supported filters and configure it. Moreover, this window displays all the evaluation metrics of the running filters.

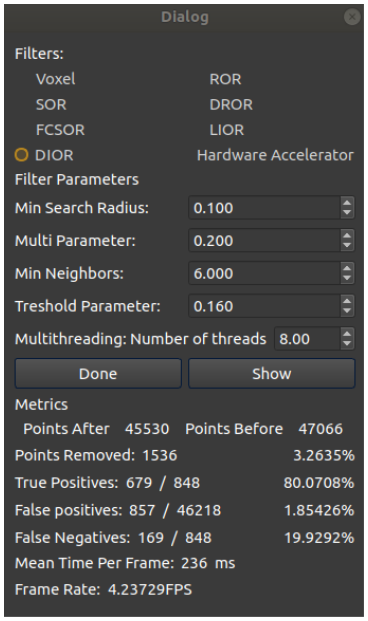


Figure 4.23: ALFA-DVC filter window.

Interface Between ALFA-Pd and ALFA-DVC

The ALFA-Pd embbedded platform interfaces and communicates with the ALFA-DVC through a ROS environment. As depicted in Figure 4.24, each module contains two publishers and two subscribers, which are responsible for transmitting point clouds, filter configurations, and performance metrics. The transmission of point clouds can be triggered in two ways: when the user presses the corresponding button in ALFA-DVC or when another ROS node publishes data into the specified ROS topic. Then, ALFA-Pd processes the point clouds received and responds with the filtered point cloud and metrics gathered during the process. Finally, ALFA-DVC displays the achieved results.

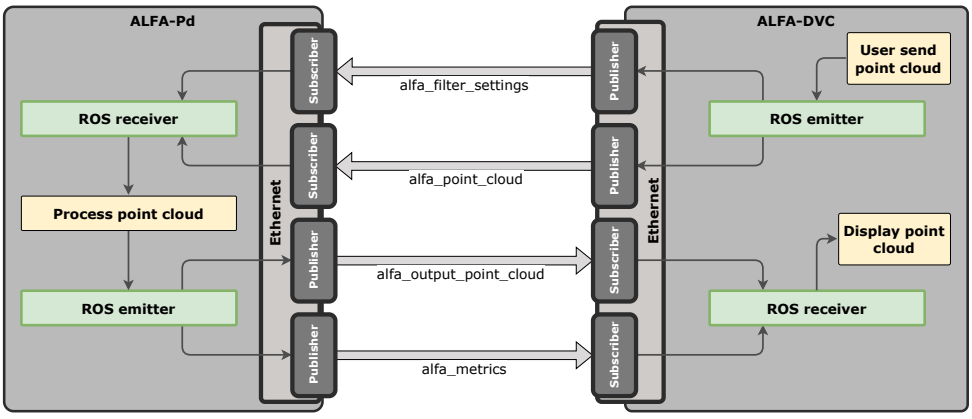


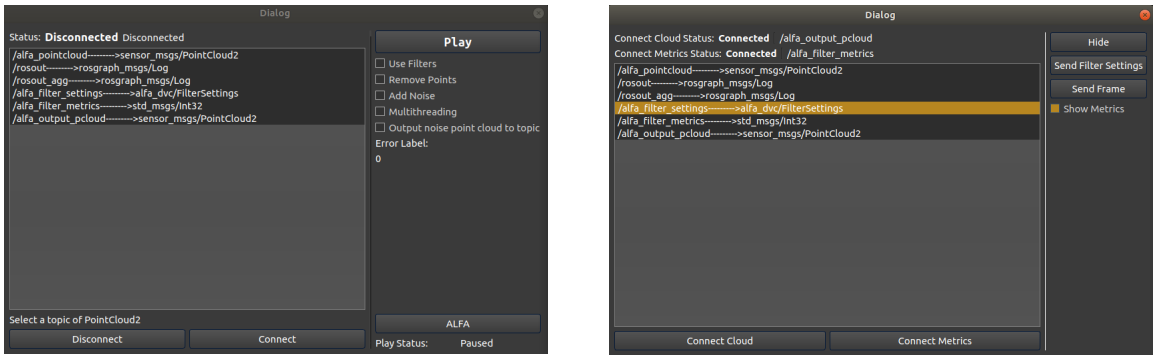
Figure 4.24: ROS-based architecture used in the ALFA-Pd

The ALFA-Pd's filters configuration are sent to a ROS topic named *alfa_filter_settings* through a custom ROS message type, allowing ALFA-DVC and other applications to configure ALFA-Pd as long as they comply

with the established message format. The point cloud data is sent to *alfa_output_point_cloud* using the Pointcloud2 format, which is a ROS point cloud message format used to represent arbitrary n-D (n-dimensional) data. Point values are of any primitive data types (int, float, double, etc.), and the message can be specified with height and width values, giving the data a 2D structure.

The ALFA-DVC tool has two dedicated subscribers to read the topics where ALFA-Pd publishes all data. Moreover, the user interface is used to choose and connect to these topics. Upon receiving a point cloud from a ROS topic, an event triggers the display of the point cloud. Additionally, receiving the metrics information, triggers an independent event, in which ALFA-DVC compares the sent and received point clouds based on the information provided by the box system. Finally, the filter window displays the obtained metrics.

ALFA-DVC interfaces with the ROS environment through two different windows, depicted in Figure 4.25. The ROS interface window, shown in Figure 4.25a, allows the user to define the ALFA-DVC workflow when point clouds are retrieved from the ROS environment. The user must initially connect to a topic having messages in the Pointcloud2 format to acquire the point clouds. Additionally, it is possible to define the actions that execute after receiving a point cloud frame, which includes: filtering the point cloud using the ALFA-DVC's built-in weather denoising filter, removing all points inside the noise removal boxes, or injecting noise into the point cloud through the noise boxes.



(a) ROS interface window.

(b) ALFA window.

Figure 4.25: ALFA-DVC ROS windows.

5. Evaluation and Results

This Chapter presents an extensive benchmarking and comparative analysis of the proposed systems and algorithm. The Chapter addresses the performance evaluation performed on the software versions of state-of-art algorithms, followed by the ALFA-Pd's hardware-accelerated versions validation and evaluation results. The denoising algorithms were evaluated using the following metrics: (i) Points Removed (PR), Equation 5.1; (ii) True Positives (TP), Equation 5.2; (iii) False Positives (FP), Equation 5.3; (iv) False Negatives (FN), Equation 5.4; (v) Frame Processing Time (FPT); and (vi) Frames Per Second (FPS). This approach ensures that all algorithms are compared fairly in terms of time-related performance and metrics relevant to the denoising of LiDAR point clouds. The ALFA-DVC's box system assessed the point cloud denoising related metrics. For visualization purposes, the input and output of each algorithm are also shown through ALFA-DVC. Since the box system occupies a lot of the visual space, the boxes are hidden in the following screenshots. Concerning the data utilized to conduct the testing and analysis, all evaluation tests were conducted using a point cloud from a publicly available dataset obtained with a Velodyne Puck (VLP-16) in a real-world heavy snowfall environment. The LiDAR sensor provides, on average, of 17098 points per frame in the used dataset.

$$PR = Input\ Points - Output\ Points \tag{5.1}$$

$$TP = \frac{Filtered\ Noise\ Points\ In\ Noise\ Areas}{Total\ Labeled\ Noise\ Points} \tag{5.2}$$

$$FP = \frac{Filtered\ Noise\ Points\ In\ Non\ Noise\ Areas}{Total\ Labeled\ Non\ Noise\ Points} \tag{5.3}$$

$$FN = \frac{Unfiltered\ Noise\ Points\ In\ Noise\ Areas}{Total\ Labeled\ Noise\ Points} \tag{5.4}$$

5.1 Evaluation of Software-based Denoising Algorithms

The software-based denoising techniques were evaluated using the ALFA-Pd embedded ROS node to run the denoising filters and ALFA-DVC to collect the filtering results and compute the performance

metrics. Since the objective of this Chapter is to demonstrate the performance of state-of-the-art algorithms utilizing an embedded environment, the filters were executed in the embedded hardcore processor. Some algorithms were improved with multithreading, which were developed to ensure a fair comparison with their hardware-accelerated implementation. Thus, performance is analyzed on native implementation but also with software-accelerated implementation, emphasizing the advantages of hardware-accelerated implementation.

5.1.1 System Configuration

All software was executed on an embedded Linux platform (4.19.0-Xilinx-v2019.2) that includes the PCL library (version 1.8.1-r0) and a ROS environment (Ros1 melodic distribution). The software is supported by the hardware platform, discussed in Section 3.1, equipped with a 1.2 GHz CPU and 2 GB of 535 MHz DDR4 memory. Additionally, because the CPU Arm Cortex-A53 has four processing cores, the multithread configurations are set to use four threads to optimize the throughput of filtered points per thread. Only the time regarding the algorithm output values is considered to calculate the FPT metric, discarding the time consumed ROS transactions since these do not concern the algorithms themselves.

Table 5.1: Filter parameters.

Algorithm	Parameter	Value
SOR	Number of k neighbors	4
	Standard deviation multiplier	0.9
ROR	Searching radius	0.5
	Min. number of neighbors	5
DROR	Min. searching radius	0.1
	Min. number of neighbors	30
	Angular resolution	0.3
	Radius multiplier	0.9
LIOR	Searching radius	0.5
	Min. number of neighbors	5
	Intensity threshold	4
DIOR	Min. searching radius	0.1
	Angular resolution	0.3
	Radius multiplier	0.9
	Min. number of neighbors	30
	Intensity threshold	4

Table 5.1 lists the denoising methods filter settings used during this evaluation. Since the algorithms' performance is highly dependent on their setups, all tests were conducted with these values. Therefore, the hardware-accelerators parameters were set to the most equivalent parameters to minimize the influence of configurations on this evaluation.

5.1.2 Voxel-Grid

The VG filter removes noise from a point cloud by downsampling the point cloud data. Furthermore, VG-based filters are the most time-efficient because of their low complexity. The before/after comparison of the VG filter is shown in Figure 5.1. In the original frame, demonstrated on the left side of the figure, the noise created by snow is visible. Contrarily, the right side displays the filtered point cloud. On the right side is illustrated the output point cloud of this filter.

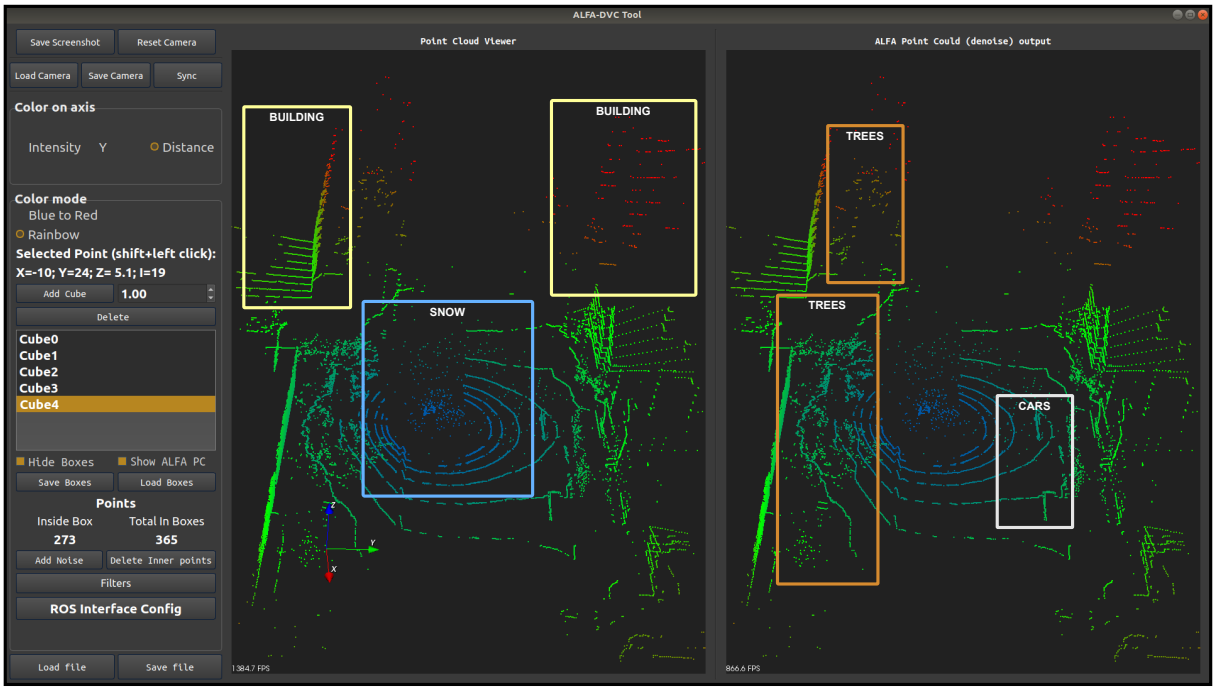


Figure 5.1: Software VG filter output.

VG filters work by establishing three-dimensional boxes (a voxel grid) in the three-dimensional space of the point cloud and then selecting a point to represent the other points within each voxel. Thus, the dense point cloud zones are the most influenced by this filter, but the dense zones are so dense that they appear to have no alteration. However, while evaluating the ALFA-DVC metrics, it is feasible to discard the difference between the original and filtered point clouds.

Table 5.2: Software VG performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
Voxel Grid	28.8%	12%	29%	87%	6	166

Table 5.2 depicts the performance metrics obtained using the box system in ALFA-DVC. The VG filter took 6 ms to finish its processing, achieving a frame rate of 166 FPS. However, the VG filter removed 28% of the points present inside the point clouds. Additionally, the VG filter only removed 12% of the noise-labeled points and wrongly classified 87% of the points removed from the point cloud. In short, this filter

demonstrates lower denoising accuracy when compared to other denoising filters, despite offering good performance metrics.

5.1.3 Statistical Outlier Removal

The SOR filter classifies points as outliers based on the neighbor information. It is one of the most mature state-of-art filters, and since the PCL provides a direct implementation of the algorithm, it was employed recurring to this library for the evaluation. Figure 5.2 depicts the processing of the point cloud corrupted by snow and the output after applying the SOR filter. Visually, it is possible to see that this filter did remove most of the labeled noise points, even though some details of buildings, trees, and road artifacts were lost. With this filter, it is possible to observe that the effect of the sparsity in the point cloud is visible since the most distant objects, shown by the red color, and the tree's leaves were deleted. The picture is zoomed in to aid in visualization, which hides the filtering performance in distant objects.

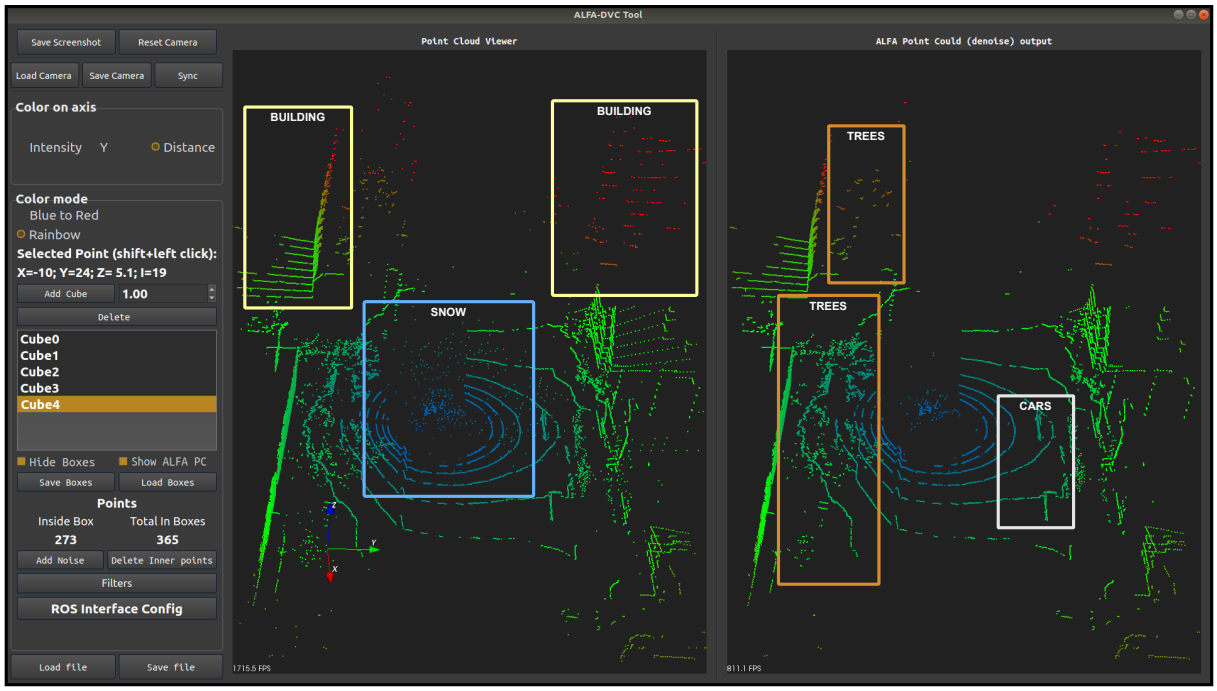


Figure 5.2: Software SOR filter output.

Table 5.3 summarizes the performance metrics acquired using the ALFA-DVC box's system. The SOR filter needed an average of 175 ms to complete processing a frame, achieving up to 5 FPS. However, most rotor-based LiDAR sensors generate point cloud frames at a rate of 10 FPS. Thus, requiring more than 175 ms to process, SOR starts introducing delays in the execution flow.

Table 5.3: Software SOR filter performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
SOR	7%	56%	5%	44%	175	5.16

The SOR filter eliminated an average of 7% of all points in the point cloud, in which, of the labeled noise points, more than half were correctly classified as outliers. However, SOR wrongly classified 5% of the point cloud as outliers. To conclude, SOR provides a simple method for weather denoising at the cost of some influence from the point cloud's sparsity. Moreover, this filter may be acceptable in environments with low noise levels, despite lacking a true-positive ratio at high levels of noise.

5.1.4 Fast Cluster Statistical Outlier Removal

The FCSOR filter is a combination of SOR and a voxel step that significantly decreases the processing time for a point cloud frame. Since FCSOR was created for mapping proposals, the author did not provide any findings for autonomous applications. The before/after comparison using the FCSOR filter is shown in Figure 5.3. Being a merge of VG and SOR, it is possible to see their similarities. The downsampling is visible by zooming in on the image, and like with SOR, a considerable loss of information is observable in distant objects and buildings. Nonetheless, a very significant noise caused by the snowfall was removed close to the vehicle, in the point cloud center.

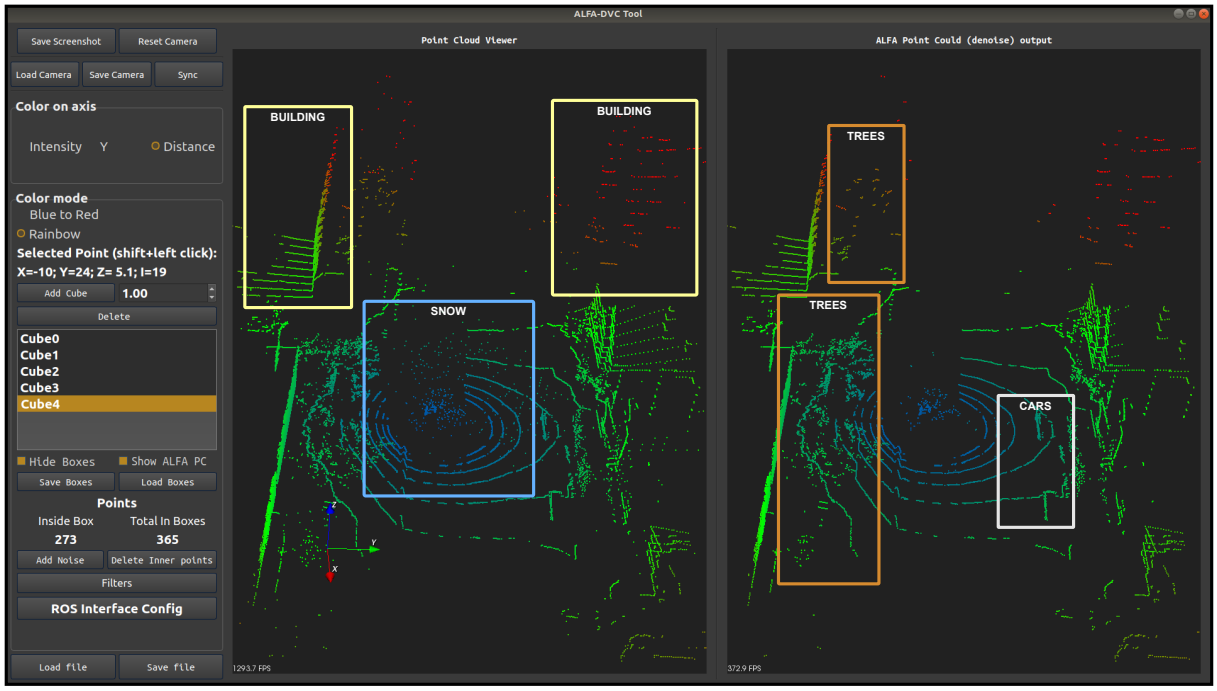


Figure 5.3: Software FCSOR filter output.

Table 5.4 depicts the performance metrics obtained using the box system in ALFA-DVC. FCSOR achieved an average frame processing time of 117 ms, enabling the processing of almost 9 FPS. Since SOR filters go through all the points that compose a point cloud, by reducing the number of points, with the VG step, FCSOR decreases the processing time required per frame when compared to native SOR. However, even with significant speed enhancements, it cannot perform real-time point cloud processing in the embedded system.

Table 5.4: Software FCSOR filter performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
FCSOR	34%	59%	34%	40%	117	8.5

FCSOR filter removed 34% of the points in the point cloud and over 60% of the labeled noise points, increasing the performance compared to SOR. However, due to the VG step, the FP ratio increases to 34%, indicating a significant removal of points in dense zones of the point cloud. The denser the point cluster, the more points are removed by the VG step. FCSOR presents a fast SOR solution, ideal for applications where downsampling the point cloud is not an issue. Regarding the noise removed, FCSOR can have a good performance in ideal weather conditions, but in extreme conditions like an intense downfall, FCSOR shows some lack of performance.

5.1.5 Radius Outlier Removal

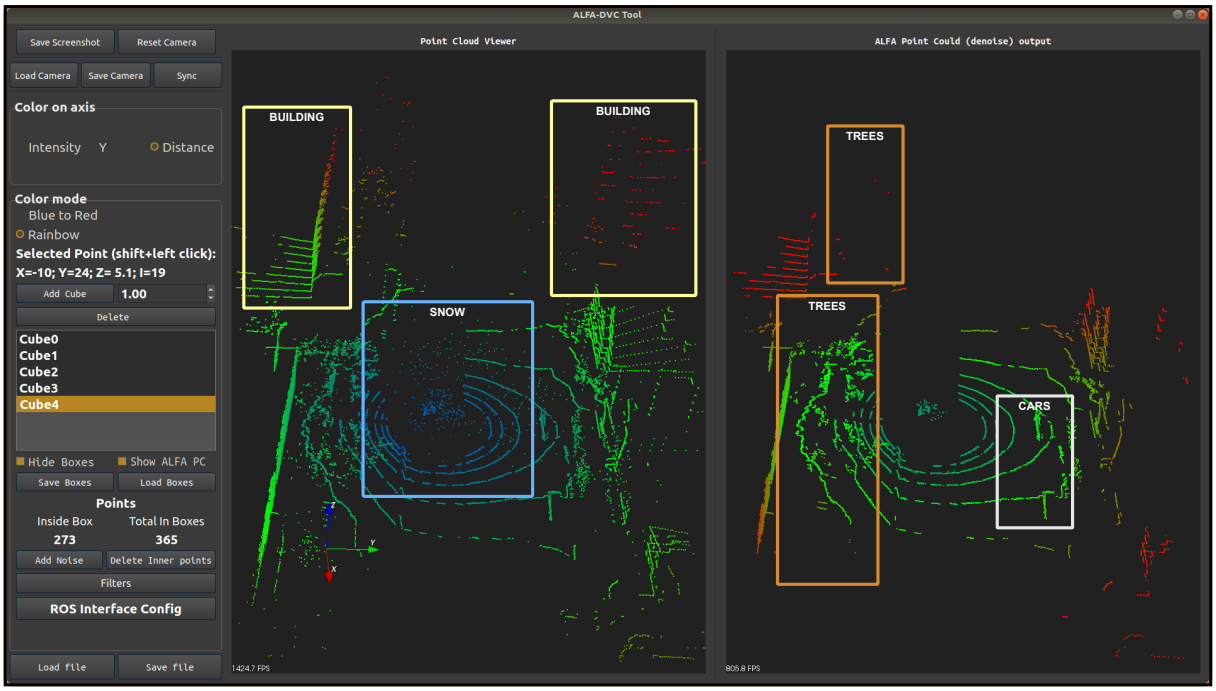


Figure 5.4: Software ROR filter output.

The ROR filter eliminates points with a small number of neighbors inside a given radius. It is one of the most mature state-of-art filters, and it has a direct implementation in the PCL library. ROR is typically used in point clouds that are not sparse, which presents a disadvantage when considering the sparse nature of LiDAR point clouds. The before/after comparison using the ROR is depicted in Figure 5.4. It is possible to see that ROR eliminated most of the labeled noise points, leaving just the cluster of snow on top of the sensor in the point cloud's center. Because of RORs fixed radius, it cannot correctly classify objects located at a higher distance from the sensor, thus removing structures and relevant information. It is

possible to conclude that ROR did remove almost all the noise present in the point cloud, at the expense of essential data in distant objects. The tree leaves were also significantly deleted by ROR, which are crucial for high-level object classification algorithms.

Table 5.5 summarizes the performance metrics acquired using ALFA-DVC's box system. The ROR filter processes a point cloud frame in an average of 180 ms, which results to an average of 5.5 FPS when executing on an embedded device. Like other filters, ROR delays the execution flow by operating at a frame rate of less than 10 FPS. Subsequently, ALFA-Pd must reject unprocessed frames to ensure that the most recent point cloud frame gets processed. ROR deleted around 19% of the point cloud total number of points, and 75% of the labeled noise points, leaving just 24% unclassified. However, ROR incorrectly identified 24% of the points deleted, which corroborates the visual analysis of the buildings and trees removed from the point cloud. Globally, ROR is a low-complexity method for outlier elimination that performs well in adverse weather conditions and in applications where distant targets are irrelevant. Since some autonomous applications rely on LiDAR to detect distant objects, ROR is incompatible with such applications.

Table 5.5: Software ROR filter performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
ROR	19%	75%	18%	24%	180	5.50

5.1.6 Dynamic Radius Outlier Removal

As with ROR, the DROR filter eliminates points with few neighbors in a radius surrounding them. However, DROR uses a dynamic radius to address the sparsity of 3D LiDAR point clouds instead of a fixed radius. Two versions of DROR were evaluated: the original, derived from author-provided pseudocode, and the multithreaded version, created in this dissertation. The before/after comparison using the DROR filter is shown in Figure 5.5. The visual output of the original and multithreaded versions was identical, so only one figure is used to represent them. To the naked eye, DROR eliminated most of the labeled snow noise except the cluster of points near the vehicle that results from snow accumulation in front of the sensor. Additionally, DROR retained information in distant objects such as trees and buildings eliminating nearly all noise from the point cloud, demonstrating a very capable denoising algorithm. Furthermore, the tree leaves, which resemble noise, were preserved, which did not happen in other filters, making this an advantage for DROR.

Table 5.6 summarizes the performance metrics acquired using ALFA-DVC's box system. DROR requires an average of 2730 ms for each frame due to the search radius calculus and the neighbor search utilizing that radius. On the other hand, the multithreaded version can process an entire frame in around 1193 ms, which represents an improvement of 56% compared to the original version. Despite this improvement, the multithreaded version can only achieve 0.83 FPS, which is unsuited for real-time applications.

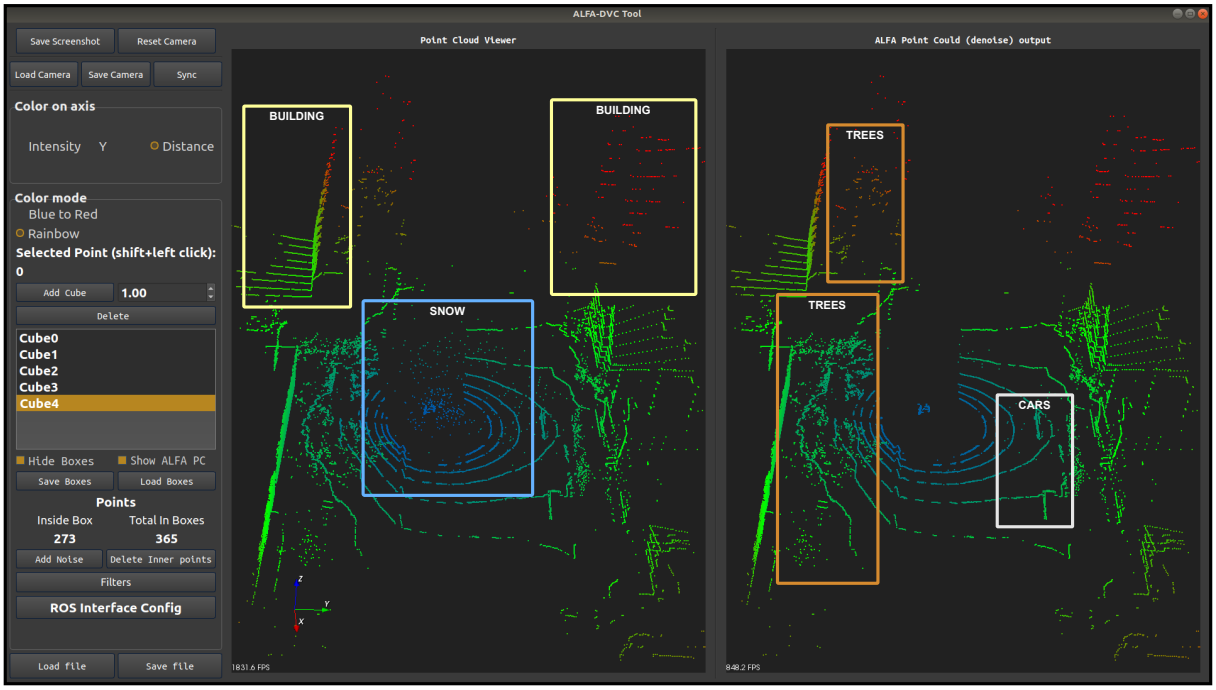


Figure 5.5: Software DROR filter output.

The long processing times are mainly due to the PCL's method for neighbor search since it is necessary to perform the search radius calculation for each point.

Table 5.6: Software DROR filter performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
DROR 1T	2%	87%	1%	13%	2730	0,36
DROR 4T	2%	87%	1%	13%	1193	0.83

DROR deleted 2% of the point cloud points, including 87% of the identified noise points. Apart from the identified noise points, DROR incorrectly classified roughly 1% of non-labeled noise points as outliers, demonstrating remarkable effectiveness in removing noise in sparse point clouds. Additionally, because the ALFA-DVC box system estimates the number of noise points, the 1% missing value may be lower in reality. Another significant advantage is that DROR can remove almost all weather noise without eliminating important information, even under severe conditions. However, DROR has a significant performance disadvantage for autonomous applications due to its inability to satisfy the time limits imposed by those applications.

5.1.7 Low-Intensity Outlier Removal

The LIOR filter can identify points as inliers based on the intensity levels. If a particular point does not satisfy the intensity requirements, a second step is applied using a ROR technique. To address the issue caused by the sparsity in point clouds produced by 3D LiDAR sensors, LIOR filters points with a

distance lower than 71 meters. However, this parameter was not implemented in the versions present in this work. The rest of the algorithms were implemented based on the description provided by the author and were further accelerated using a multithread configuration. Both implementations were tested to analyze the impact of parallelizing the processing flow. The before/after comparison using the LIOR filter is shown in Figure 5.6. As with DROR, multithreading solely impacts the time-based metrics, and hence only one picture is displayed to describe the output. Although the filter successfully removes most of the labeled noise, denser noise created by severe snowfall is still incorrectly recognized as an inlier by the filter. Additionally, because LIOR has a limited radius, as the distance between points grows for far objects, the information about these distant items is lost. It is possible to observe that trees leaves are also classified as outliers, which can cause further problems in high-level applications that rely on the point cloud for the object recognition and classification tasks.

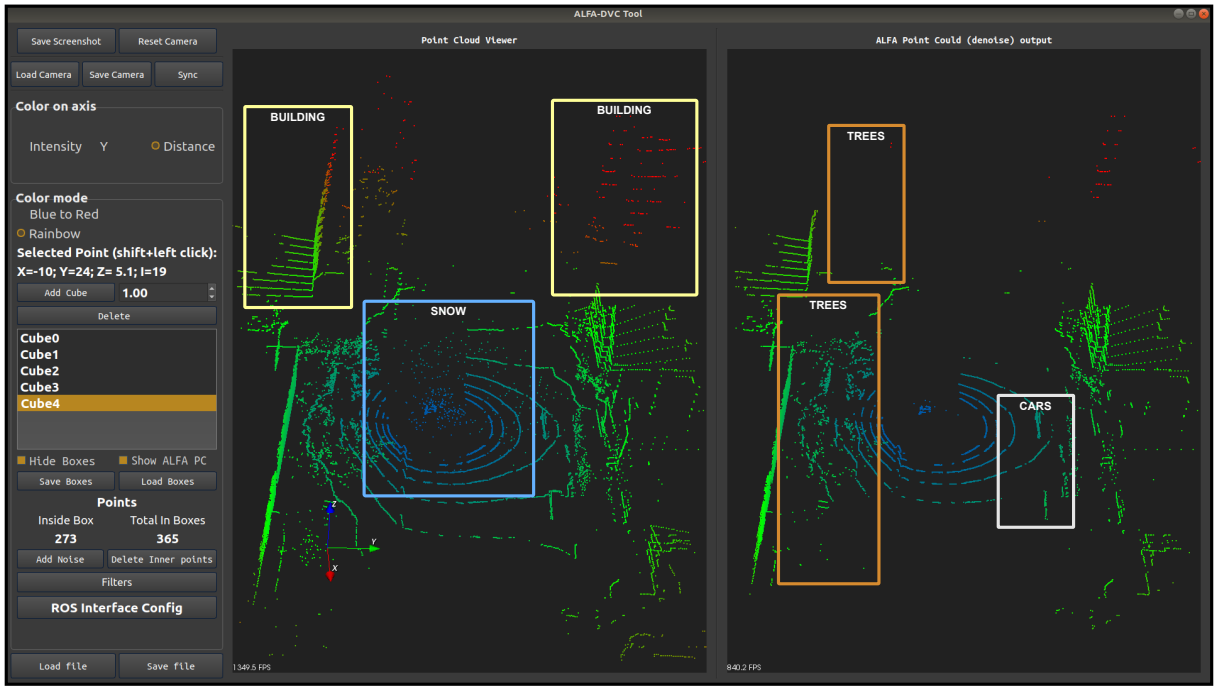


Figure 5.6: Software LIOR filter output.

Table 5.7 summarizes the performance metrics acquired using ALFA-DVC's box system. The single-threaded version requires an average of 173 ms per frame to execute. On the other hand, the multithread configuration reduces processing time to 109 ms, representing a reduction of 37% over the single thread version. Given these frame processing durations, the single-threaded algorithm can generate 5.7 FPS, and the multithreaded accelerated version can obtain a maximum frame rate of 9 FPS.

Table 5.7: Software LIOR filter performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
LIOR 1T	11%	74%	10%	25%	173	5,7
LIOR 4T	11%	74%	10%	25%	109	9,17

LIOR removed 11% of all point cloud points, demonstrating its similarity to ROR. LIOR deleted around 74% of the labeled snow points but categorized 10% of the non-noise points incorrectly. In general, LIOR is a fast-running algorithm with strong denoising capabilities. LIOR is an efficient technique in severe weather circumstances, especially when an ROI is applied, as seen in the authors experiments. However, LIOR performs poorly in adverse weather circumstances compared to other state-of-the-art algorithms.

5.1.8 Dynamic low-Intensity Outlier Removal

The DIOR filter is a novel to weather denoising proposed by this dissertation. To compensate for the sparsity of the point cloud, DIOR employs a dynamic search radius, which eliminates points with few neighbors in a radius surrounding them. DIOR was evaluated in the same single thread/multithread configurations as LIOR and DROR, with the workload divided among four threads to conduct a fair comparison of DIOR and other state-of-the-art algorithms. The before/after comparison using the DIOR filter is shown in Figure 5.7. Only one figure depicts the result since the original and multithreading visual output was identical. Visually, DIOR removes all labeled noise points while retaining information about distant objects and tree leaves. When comparing the results obtained with DIOR and DROR, it is possible to see the similarities of both algorithms. Overall, DIOR presents a competent filter for denoising noise caused by adverse weather and does not require the use of an ROI to retain point cloud data, as LIOR does.

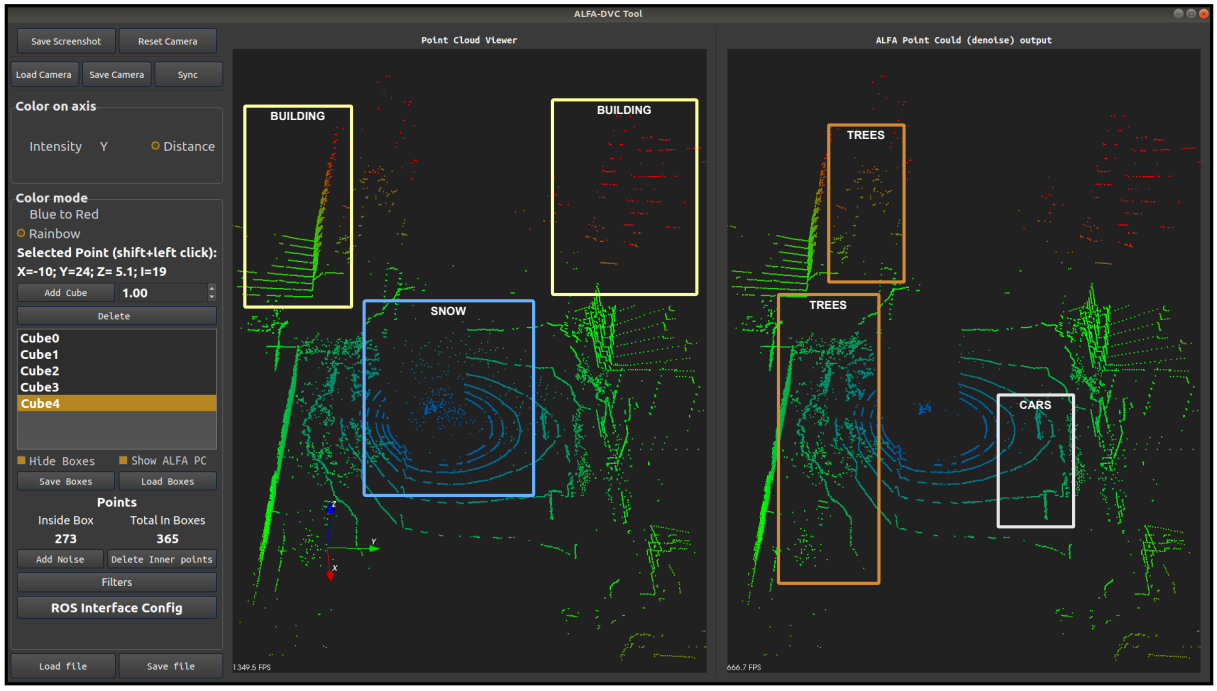


Figure 5.7: Software DIOR filter output.

Table 5.8 depicts the performance metrics obtained using the box system in ALFA-DVC. Firstly, DIOR achieves an average FPT of 1009 ms without utilizing multithreading. Using multithreading resulted in the filter's processing time being lowered to an average of 442 ms per frame, representing an enhancement

of 56%. Therefore, the original approach can output around 1 FPS with the measured frame processing time, whereas the multithread accelerated version can generate 2.26 FPS, meaning no version of DIOR can keep up with the throughput of most LiDAR sensors.

Table 5.8: Software DIOR filter performance.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
DIOR 1T	2%	87%	0%	14%	1009	0,99
DIOR 4T	2%	87%	0%	14%	442	2,26

Regarding DIOR's denoising efficiency, the filter deleted 2% of all the points inside the point cloud, deleting around 87% of the noise labeled points, with approximatively 0% false positives. DIOR is highly efficient in removing noise points, missing just 14% of label points due to the noise's great density when close to the vehicle. Moreover, this filter produced results comparable to those obtained with DROR but with a better false-positive ratio. By combining the LIOR and DROR concepts, DIOR can attain the high accuracy of DROR while significantly shortening the processing time necessary to analyze a point cloud frame. Despite its good denoising performance, in its software versions, DIOR is still not suitable for automotive applications because of its high frame processing times.

5.1.9 Discussion

With the software versions of the algorithms analyzed (depicted in Table 5.9), some conclusions and comparisons can be made. Despite having the fastest processing time, the VG filter presents the poorest denoising performance metrics, with an FN of nearly 87%. Regarding DROR, it is one of the best performing filters in terms of denoising capabilities at the cost of real-time performance. On the other hand, SOR and FCSOR present TP rates between 55% and 60%, with an FN of nearly 40%. LIOR, which uses ROR principles, provides good accuracy results while being the fastest algorithm of the tested. Lastly, DIOR achieves great results regarding its PR, around 2%, FN of 14%, and FP as low as virtually 0%, at the cost of higher FPT.

Table 5.9: Evaluation of software-only denoising algorithms.

Algorithm	PR	TP	FP	FN	FPT (ms)	FPS
Voxel-Grid	28.8%	12%	29%	87%	6	166
ROR	19%	75%	18%	24%	180	5.50
SOR	7%	56%	5%	44%	175	5.71
DROR	2%	87%	1%	13%	1193	0.83
FCSOR	34%	59%	34%	40%	117	8.5
LIOR	11%	74%	10%	24%	109	9.17
DIOR	2%	85%	0%	14%	442	2.26

The ALFA-Pd software-only node results in long processing times for filtering due to the hardware platform's limited resources, such as the CPU and DDR memory speed. However, because the platform

supports hardware acceleration, the most promising algorithms, DROR, LIOR, and DIOR, were accelerated. The main objective of porting some functionalities to hardware is to improve FPT and FPS without impairing the algorithm's classification efficiency.

5.2 Hardware-Accelerated Denoising Algorithms

ALFA-Pd features several hardware accelerators to improve the real-time performance of the algorithms that have the best denoising metrics. The following Section demonstrates the results of their evaluation in both hardware implementation (BRAM and DDR). The assessment was also done using the ALFA-DVC tool, reducing testing variables.

5.2.1 Dynamic Radius Outlier Removal

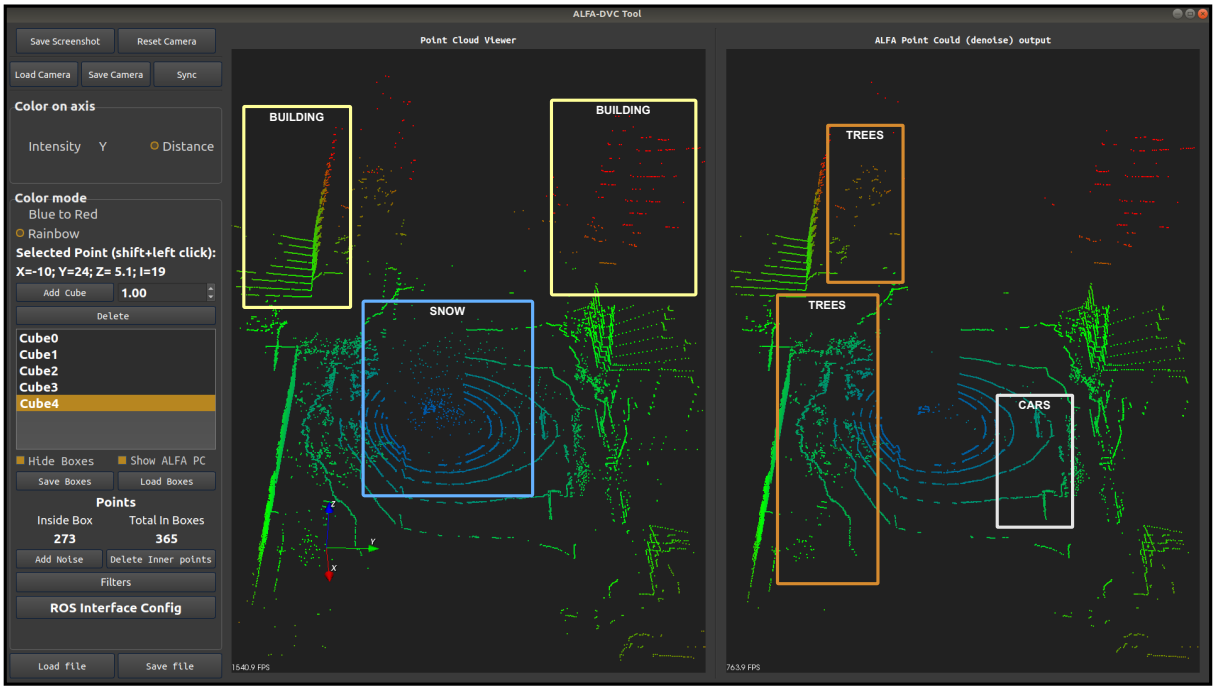


Figure 5.8: Hardware-accelerated DROR filter output.

Due to the high potential attained in early software assessment, DROR was one of the filters implemented in hardware. The significant disadvantage of DROR compared with the remaining was that it did not fulfill the time limitations associated with autonomous applications, requiring hardware acceleration. The before/after comparison using the DROR filter is shown in Figure 5.8. The result is visually identical to the software implementation, where the hardware-accelerated version was capable of eliminating the majority of the identified noise points. Additionally, the buildings and tree points were kept unchanged, indicating that the hardware version is performing correctly as the point's distance does not influence its performance.

BRAM Implementation

ALFA-Pd offers a variety of hardware combinations, which affect the algorithm's performance. Thus, several hardware configurations were evaluated, gradually increasing the number of PCs from two to 32 and maintaining the NF number limited to two since the BRAM can only acquire two points each clock cycle. Table 5.10 depicts the results obtained by evaluating the DROR hardware BRAM implementation. The base implementation, which used two PCs, averaged around 689 ms to analyze an entire frame. Nonetheless, when the number of PCs doubles, the time necessary to process a point cloud reduces to half, which represents an almost linear improvement. At 32 PC units, the hardware-accelerated version can process an entire frame in 40 ms, representing 27.77 FPS. Additionally, to achieve real-time filtering, the throughput of the filter needs to be higher than 10 FPS, which DROR can achieve using 16 PCs. Despite having slightly different filter configurations (rounding, floating-point handling), the hardware-accelerated BRAM implementation still presents filtering results similar to its software counterpart. This version deleted around 4% of the point cloud, with 87% of the noise points eliminated and only 3% false positives.

Table 5.10: Hardware-accelerated DROR filter BRAM performance.

Algorithm	PCs	PR	TP	FP	FN	FPT (ms)	FPS
DROR	2	5.8%	95%	3%	5.4%	689	1.45
	4	5.3%	95%	3%	5.5%	350	2.8
	8	5.6%	93.2%	3.4%	6.8%	170	5.89
	16	5.35%	92%	3%	7%	84	11.9
	32	4%	87%	3%	13%	40	27.77

DDR Implementation

The DDR implementation performs similarly to the BRAM implementation but with an increase in the FPT metric. Notwithstanding the fastest implementation of this version achieving 18.1 FPS, the 16 PC configuration is still sufficient to handle most 3D LiDAR sensor's throughput. In general, the DROR hardware implementation meets the requirements for usage in autonomous applications by providing a high true positive ratio while still complying with time constraints.

Table 5.11: Hardware-accelerated DROR filter DDR performance.

Algorithm	PCs	PR	TP	FP	FN	FPT (ms)	FPS
DROR	2	6%	92%	4.3%	7.6%	1173	0.85
	4	5.9%	92%	4.2%	7.8%	545	1.84
	8	5.6%	90%	3.75%	9.4%	220	4.5
	16	5.35%	92%	3.4%	7.9%	84	11.9
	32	5.25%	91%	3.6%	8.3%	55	18.1

5.2.2 Low-Intensity Outlier Removal

LIOR was one of the filters implemented in hardware due to the high potential shown in the early software evaluation. LIOR achieved significant results in efficiency metrics, alongside outstanding performance, achieving almost 10 FPS. Because of this, ALFA-Pd offers a hardware-accelerated version of this algorithm. The before/after comparison using the LIOR filter is shown in Figure 5.9. The hardware version of LIOR has many similarities to its software version, and although the filter eliminated a significant portion of the labeled noise, it is clear that some information about distant objects was lost. The most severely damaged areas of incorrect classification were the tree leaves, which were deleted from the point cloud. Nevertheless, compared to the software implementation, it is possible to see that the number of relevant information lost is smaller, showing a minor improvement over it.

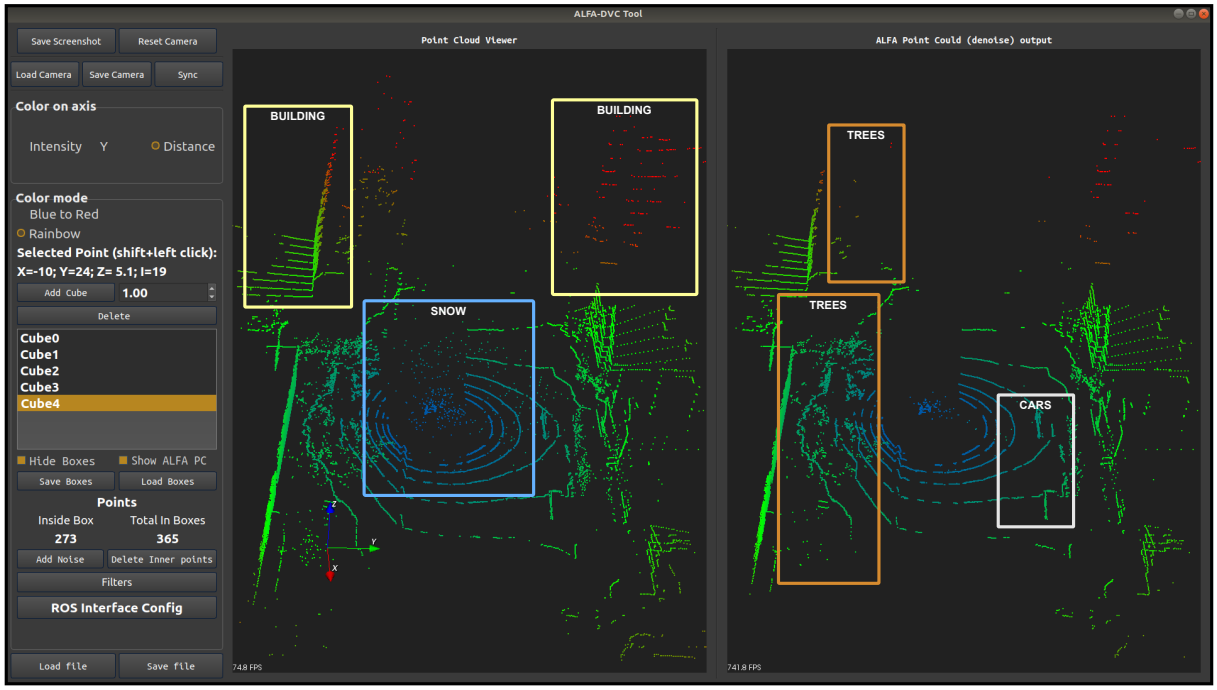


Figure 5.9: Hardware-accelerated LIOR filter output.

BRAM Implementation

The BRAM implementation proposes a faster execution version of LIOR, which tries to achieve real-time filtering. As with all the hardware-accelerated algorithms, it was evaluated with different PCs configurations to see their performance impact. Table 5.12 depicts the results obtained through evaluating the LIOR hardware BRAM implementation. When using 2 PCs, the filter took 631 ms to process an entire frame. As expected, by increasing the number of PCs, the time required per frame also decreases, making LIOR capable of processing 33 FPS in the 32 PCs configuration. It is essential to understand that the only constraint on the number of PCs is the available resources of the platform in use. Nevertheless, the

hardware BRAM LIOR version already satisfies the time requirements with 16 PCs. Regarding efficiency, LIOR deleted an average of 9% of all points in the point cloud. The filter accurately categorized 68% of the noise points as inliers. In terms of FP, LIOR incorrectly categorized 7% of the points as outliers, demonstrating the algorithm's inefficiency when applied to sparse point clouds.

Table 5.12: Hardware-accelerated LIOR filter BRAM performance.

Algorithm	PCs	PR	TP	FP	FN	FPT (ms)	FPS
LIOR	2	11%	73%	9%	26%	631	1.58
	4	10.65%	72.05%	8.9%	28%	308	5.31
	8	9%	73%	8.9%	30%	143	6.9
	16	9%	71%	7.6%	30%	56	19.6
	32	9%	68%	6.9%	34%	30	33.3

DDR Implementation

Non time-related performance metrics were quite similar to those obtained while analyzing the BRAM implementation, as seen in Table 5.13. While the DDR implementation has slower execution speeds, it still complies to the time constraints imposed by the throughput of most 3D LiDAR sensors. The 32 PC configuration is capable of delivering 20 processed FPS. To summarize, hardware-accelerated LIOR enhanced the old version's time performance.

Table 5.13: Hardware-accelerated LIOR filter DDR performance.

Algorithm	PCs	PR	TP	FP	FN	FPT (ms)	FPS
LIOR	2	14%	70%	12%	30%	970	1.03
	4	10.81%	67%	9.25%	33%	383	2.61
	8	11.48%	68%	10.25%	32%	150	6.71
	16	11.56%	68%	11.3%	31%	56	19.6
	32	12%	68%	12.4%	31%	49	20.408

5.2.3 Dynamic low-Intensity Outlier Removal

DIOR, the dissertation's proposed algorithm, was accelerated in hardware due to the good results obtained in the software tests. Despite its high efficiency in removing weather noise from 3D LiDAR point clouds, it demonstrated the need for acceleration, as it did not fulfill the time limitations for autonomous applications. A before/after comparison using the DIOR filter is shown in Figure 5.10, created using a 32 PC setup. As with other hardware algorithms, the hardware version of DIOR produces similar visual results compared to the software version. This filter reduced most snow-generated noise with changes between the original and filtered images visible to the human eye. Furthermore, the filter's efficiency is quite good due to the features inherited from DROR. The algorithm left faraway objects untouched, while tree leaves, which are the most difficult items to identify using outlier-based approaches, were barely eliminated.

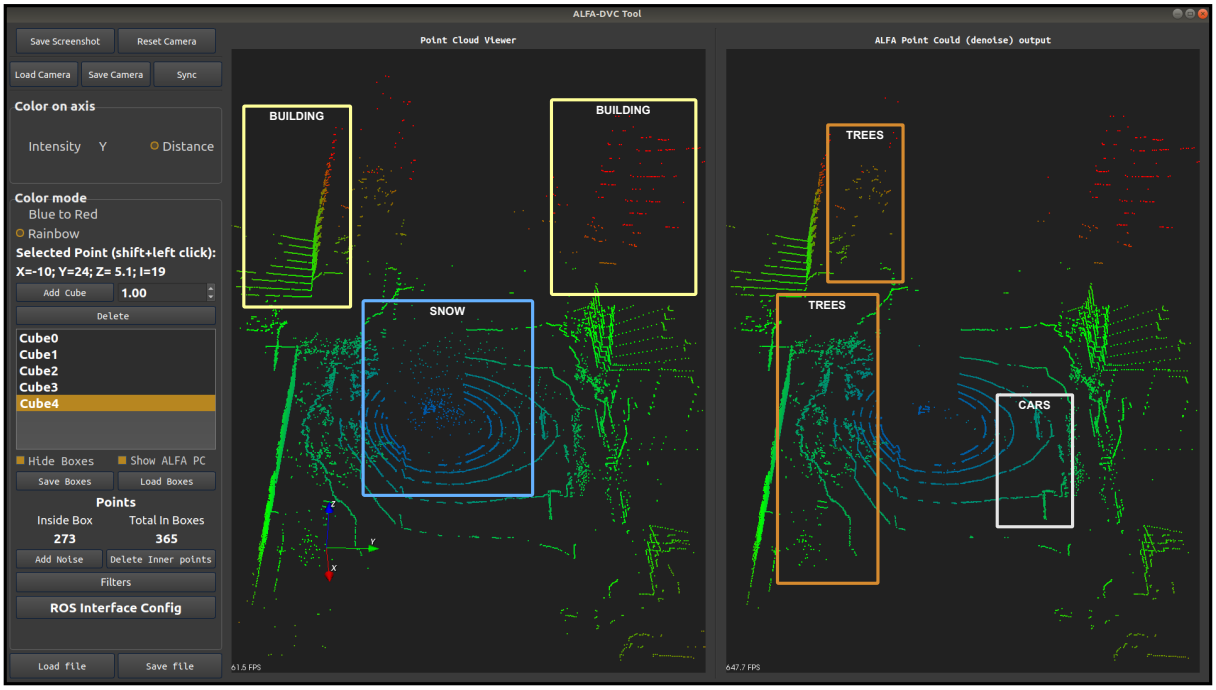


Figure 5.10: Hardware-accelerated DIOR filter output.

BRAM Implementation

The BRAM implementation of DIOR features a quicker execution speed, and as with the other hardware-accelerated algorithms, it was analyzed using various PC setups to determine their effect on the filter's overall performance. Table 5.14 summarizes the performance metrics acquired using the ALFA-DVC box's system. LIOR processes a point cloud frame in 476 ms using the base implementation. Naturally, by increasing the number of PCs, DIOR reached a frame processing time of 30 ms using the fastest implementation. Notwithstanding the time necessary to process a frame comparable to LIOR, DIOR presents better denoising ratios. A 16 PC implementation is sufficient for real-time operation providing 19 FPS. Moreover, DIOR deleted an average of 4% of the point cloud's total points, and accurately recognized 90% of the labeled points as inliers. In terms of false positives, DIOR miscategorized just 3% as outliers, demonstrating the effectiveness of reducing weather-generated noise in adverse situations.

Table 5.14: Hardware-accelerated DIOR filter BRAM performance.

Algorithm	PCs	PR	TP	FP	FN	FPT (ms)	FPS
DIOR	2	5%	93%	3%	5.8%	476	2.1
	4	5%	93%	3%	5.6%	244	4.1
	8	5%	93%	3%	7%	124	8.1
	16	4.35%	91%	2.5%	7%	56	19.6
	32	4%	88%	2.8%	13%	30	33.3

DDR Implementation

The results obtained using the DDR implementation, depicted in Table 5.15, are very similar to those obtained with the BRAM implementation, apart from the time-related metrics, since by utilizing a more general memory architecture, the FPT of lower-end PC configurations rises when compared to the BRAM implementation. Nevertheless, on a higher count of PC modules, this difference is not that noticeable due to the high parallelization of the algorithm. With the faster execution configuration, this version achieved 30.3 FPS, which was more than sufficient for the sensor used to gather the dataset utilized. Since the methodology of finding neighbors in hardware differs from software one, the dynamic search radius does not impact the processing time. Thus, it is possible to achieve high ratios without compromising the algorithm's time requirements. To summarize, DIOR in hardware presents the good time performance of LIOR, with the excellent true positive ratio of DROR.

Table 5.15: Hardware-accelerated DIOR filter DDR performance.

Algorithm	PCs	PR	TP	FP	FN	FPT (ms)	FPS
DIOR	2	6.62%	92%	5.8%	7%	761	1.35
	4	4.9%	90%	3%	7%	365	2.74
	8	4.45%	89%	2.5%	7%	146	6.84
	16	4.35%	91%	2.5%	7%	56	19.6
	32	4%	88%	3.8%	13%	33	30.30

5.2.4 Hardware Resources

ALFA-Pd has a modular approach, allowing the user to choose the number of PCs and NF modules, which increases the parallelization of the denoising method, hence enhancing performance. However, the resources required to produce the hardware rise substantially. Thus, a study of the requirements for each PC setup was conducted. Table 5.16 summarizes the hardware resource consumption for both versions,

Table 5.16: Hardware resources.

PCs	Version	LUTS	FFs	BRAM	DSP
2	BRAM	7.79%	4.05%	83.3%	0.92%
	DDR	3.41%	2.19%	3.84%	0.57%
4	BRAM	9.01%	4.18%	83.3%	1.85%
	DDR	4.77%	2.33%	3.84%	1.15%
8	BRAM	11.96%	4.39%	83.3%	3.7%
	DDR	7.19%	2.62%	3.84%	2.31%
16	BRAM	19.09%	5.09%	83.3%	7.4%
	DDR	13.11%	3.17%	3.84%	4.63%
32	BRAM	39.30%	5.68%	83.3%	14.81%
	DDR	26.13%	3.75%	3.84%	8.68%

with the PCs ranging from 2 to 32. The three algorithms supported by the ALFA-Pd hardware implementation are all included within a PC module that may be enabled on-demand by the ALFA-DVC utility.

BRAM Implementation

To deploy the 32 PCs version, the system needs roughly 39% of the board's LUTs, 5.68% of the FFs, 83.3% of the BRAMs, and 14.81% of the available DSP blocks. The high percentage of BRAMs in use is due to pre-allocation; this does not indicate that they are constantly in use. Nonetheless, ALFA-Pd is compatible with top-of-the-line sensors capable of producing 800 000 points each frame due to this allocation. When the three algorithms analyze the selected point cloud at a frame rate of 10 Hz, 16 PCs can already deliver real-time capabilities to sensors with a 10 Hz output at the cost of a few hardware resources.

DDR Implementation

Compared to the BRAM version, the DDR version needs half the number of NFs. However, the BRAM version required more than half the resources of the DDR version in systems with more than four PCs. The latter is due to the increased complexity of this version, which escalates more rapidly as the number of PCs increases, resulting in the deployment of 32 PCs consuming about 26% of the available LUTs and 3.75% of available FFs. The number of BRAM used decreased significantly, from 83% to 3.84%, and the number of DSP also dropped to 8.68%. Nonetheless, it is possible to reach real-time with 16 PCs when using the DDR version and this dataset. Thus, ALFA-Pd presents denoising methods capable of filtering noisy point clouds without disturbing the execution flow of high-level applications.

5.3 Hardware vs Software Implementations

The efficiency in detecting and removing noise was identical in the software and the hardware versions, but it is still noticeable that the hardware version of LIOR improved the base algorithm. Such results prove the correctness of the hardware implementation of the selected algorithms since the hardware deployment accelerates the FPT metric without changing the algorithm's behavior. Increasing the number of PCs has a visible effect on the amount of time necessary to process a single point cloud frame. When the system is deployed with only two PCs, the processing time for DROR and DIOR is nearly identical to their software implementation. However, the LIOR processing time increases significantly from 109 to 631 ms. When more than two PCs are used, the results are always superior to the software-only solution, with an FPT of 40 ms for DROR and 30 ms for LIOR and DIOR when the system uses 32 PCs. The theoretical maximum number of PC and NF modules is unlimited, so if the number of PCs and NF is equal to the size of the point cloud, the frame could be processed in one clock cycle under perfect conditions. Nonetheless, using 32 PCs, the BRAM enhanced DROR's speed by about 98 times compared to the base version and 42 times compared to the multithread version. ALFA-Pd shortened the processing time required for LIOR by

3.3 times. Regarding DIOR, it showed improvements in its efficiency similar to those obtained obtained by the DROR filter, but the execution times were somewhat equal to the ones obtained by LIOR.

5.4 Closing Discussion

Comparing DROR and LIOR algorithms with DIOR and using the ALFA-Pd platform, the obtained results show that DIOR can achieve better performance ratios. Moreover, for the point cloud with 17098 points, DIOR achieved better frame processing times with an FPS rate of around 33 FPS. Regarding the hardware platform, and because the processing unit provides fewer resources when compared with the original setups used by DROR and LIOR, the ALFA framework offers the possibility to deploy hardware accelerators to achieve better performance rates at the cost of FPGA hardware resources, even when using an embedded configuration. However, newer LiDAR sensors can provide denser point clouds with millions of points to be processed. Despite increasing the accuracy of the perception system, they require more powerful resources to process the point cloud data, ideally in real-time. In the evaluation of the LIOR algorithm, the authors have used an Ouster OS-1 LiDAR with a point cloud output of around 60k points per frame. According to their results, and before applying optimizations, the processing rate performance for processing the full sensor's data was 0.16 FPS for DROR and 1.32 FPS for LIOR. To increase the performance of their filter, the authors have optimized the denoising algorithm by applying a cropbox to different ROI to reduce the number of points to be processed: Cropbox A (full data); Cropbox B (forward data); and Cropbox C (only road data). By reducing the number of points, the achieved performance speed was 9.31 FPS for Cropbox B and 10.0 FPS for Cropbox C. This way, authors could claim a frame rate that can cope with the real-time processing of the output of a LiDAR sensor which is commonly 10 Hz.

Table 5.17: Hardware-based algorithms optimization.

	Copbox A (59395 points)			**Cropbox B** (27734 points)			**Cropbox C** (25456 points)		
	DROR	**LIOR**	**DIOR**	**DROR**	**LIOR**	**DIOR**	**DROR**	**LIOR**	**DIOR**
FPT	270	76	77	78	31	31	67	28	28
FPS	3.7	13.01	12.98	12.82	32	32	15	35.6	35.7

For a fair comparison between the frame rate provided by the ALFA-Pd and LIOR's hardware setup, which includes more processing capabilities, a point cloud was processed from a Velodyne VLP-32C sensor, which can also output around 60k points per frame. Table 5.17 summarizes the obtained results when applying the same strategy of including a Cropbox over different ROI. Processing the full point cloud (Cropbox A) results in a frame rate of around 13 FPS, both for LIOR and DIOR, which copes with the requirement of processing a sensor's output of 10 Hz. Applying the Cropbox B and C reduces the number of points processed to 27734 and 25456, respectively. These results in a processing frame rate of around 32 FPS for Cropbox B and 35 FPS for Cropbox C, showing that the ALFA-Pd is able to achieve real-time capabilities even for sensors with higher output rates when ROI is applied.

6. Conclusion

With growing interest in developing fully autonomous vehicles, there is a demand to develop technologies capable of collecting accurate representations of a vehicle's surroundings. As a result, multi-sensor perception systems comprised of RADAR, Cameras, and LiDAR are becoming standard in the industry. LiDAR sensors are a relatively new addition to these multi-sensor perception systems. Nevertheless, their capacity to provide a high-resolution representation of the surrounding world in real-time has already established them as a critical component. However, the output of 3D LiDAR sensors can be affected by several noise sources, including adverse weather.

This dissertation presents a novel hardware-based technique for weather denoising, ALFA-Pd, which allows the deployment of a variety of denoising algorithms in both software and hardware, supported by the ALFA-DVC tool. This tool enables easy deployment and configuration of different weather denoising algorithms in run time, providing debug and point cloud visualization capabilities within a ROS environment. The framework's embedded nature shows how state-of-the-art methods perform in a resource-constrained environment. Furthermore, it explores how they can benefit from hardware acceleration to significantly reduce the processing time of point cloud frames while maintaining the desired accuracy metrics, such as the number of PR, TP, FP, and FN ratios.

ALFA-Pd presented two ways for optimizing state-of-the-art denoising algorithms through multithreading and hardware accelerators. The multithreading used to parallelize the execution over the four CPU cores available on the embedded platforms achieved good processing time reductions, ranging from 37% to 56%. On the other hand, using the hardware-accelerated capabilities of the platform, it was possible to further reduce those percentages up to 98.5%.

6.1 Future Work

ALFA-Pd fulfills all of the initial requirements, including the ability to select and execute several denoising algorithms with or without hardware acceleration while offering connectivity to high-level applications through a ROS environment. However, ALFA-Pd can still be enhanced in several ways, which were not integrated due to higher complexity and time constraints. The following points are possible system improvements for future work:

- **Improve hardware method for finding neighbors:** The current method for locating neighbors is a brute-force approach, in which a point is compared to all other points in the point cloud. However, there are more efficient approaches, such as FLANN, which can speed up processing even more.
- **Improve processing parallelization**: The architecture of ALFA-Pd's PC and NF modules has extensively parallelized processing. However, by increasing parallelization, for example in the memory access process, it is feasible to improve performance.
- **Integrating other types of denoising algorithms:** ALFA-Pd focuses on outlier-based algorithms, but there are other types of weather denoising methods like CNN-based, which were not explored in this dissertation.

References

[1] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Sensor Technologies for Intelligent Transportation Systems," *Sensors (Basel, Switzerland)*, vol. 18, no. 4, p. 1212, 2018.

[2] E. Marti, M. A. de Miguel, F. Garcia, and J. Perez, "A Review of Sensor Technologies for Perception in Automated Driving," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 4, pp. 94–108, 2019.

[3] B. Shahian Jahromi, T. Tulabandhula, and S. Cetin, "Real-Time Hybrid Multi-Sensor Fusion Framework for Perception in Autonomous Vehicles," *Sensors*, vol. 19, no. 20, 2019.

[4] A. S. Mohammed, A. Amamou, F. K. Ayevide, S. Kelouwani, K. Agbossou, and N. Zioui, "The Perception System of Intelligent Ground Vehicles in All Weather Conditions: A Systematic Literature Review," *Sensors*, vol. 20, no. 22, 2020.

[5] M. E. Warren, "Automotive LIDAR Technology," in *2019 Symposium on VLSI Circuits*, pp. C254–C255, 2019.

[6] Y. Li and J. Ibanez-Guzman, "Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[7] R. Roriz, J. Cabral, and T. Gomes, "Automotive LiDAR Technology: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–16, 2021.

[8] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A Survey on 3D Object Detection Methods for Autonomous Driving Applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3782–3795, 2019.

[9] X. Peng and J. Shan, "Detection and Tracking of Pedestrians Using Doppler LiDAR," *Remote Sensing*, vol. 13, no. 15, 2021.

[10] W. Huang, H. Liang, L. Lin, Z. Wang, S. Wang, B. Yu, and R. Niu, "A Fast Point Cloud Ground Segmentation Approach Based on Coarse-To-Fine Markov Random Field," *IEEE Trans. on Intell. Transp. Syst.*, pp. 1–14, 2021.

[11] R. Karlsson, D. R. Wong, K. Kawabata, S. Thompson, and N. Sakai, "Probabilistic Rainfall Estimation from Automotive Lidar," 2021.

[12] J. Jiménez, "Laser diode reliability: Crystal defects and degradation modes," *Comptes Rendus Physique*, vol. 4, 2003.

[13] W. C. Kwong, W. Y. Lin, G. C. Yang, and I. Glesk, "2-d optical-cdma modulation in automotive time-of-flight lidar systems," *International Conference on Transparent Optical Networks*, vol. 2020-July, 2020.

[14] T. Fersch, R. Weigel, and A. Koelpin, "A CDMA Modulation Technique for Automotive Time-of-Flight LiDAR Systems," *IEEE Sensors Journal*, vol. 17, no. 11, pp. 3507–3516, 2017.

[15] H. Lee, S. Kim, S. Park, Y. Jeong, H. Lee, and K. Yi, "Avm / lidar sensor based lane marking detection method for automated driving on complex urban roads," *IEEE Intelligent Vehicles Symposium, Proceedings*, 2017.

[16] M. Jokela, M. Kutila, and P. Pyykönen, "Testing and validation of automotive point-cloud sensors in adverse weather conditions," *Applied Sciences*, vol. 9, no. 11, 2019.

[17] J. R. Vargas Rivero, T. Gerbich, V. Teiluf, B. Buschardt, and J. Chen, "Weather Classification Using an Automotive LIDAR Sensor Based on Detections on Asphalt and Atmosphere," *Sensors*, vol. 20, no. 15, p. 4306, 2020.

[18] P. H. Chan, G. Dhadyalla, and V. Donzella, "A Framework to Analyze Noise Factors of Automotive Perception Sensors," *IEEE Sensors Letters*, vol. 4, no. 6, pp. 1–4, 2020.

[19] R. Heinzler, P. Schindler, J. Seekircher, W. Ritter, and W. Stork, "Weather Influence and Classification with Automotive Lidar Sensors," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2019-June, pp. 1527–1534, 2019.

[20] R. Heinzler, F. Piewak, P. Schindler, and W. Stork, "CNN-Based Lidar Point Cloud De-Noising in Adverse Weather," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2514–2521, 2020.

[21] T.-H. Sang, S. Tsai, and T. Yu, "Mitigating Effects of Uniform Fog on SPAD Lidars," *IEEE Sensors Letters*, vol. 4, no. 9, pp. 1–4, 2020.

[22] T. Yang, Y. Li, Y. Ruichek, and Z. Yan, "LaNoising: A Data-driven Approach for 903nm ToF LiDAR Performance Modeling under Fog," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10084–10091, 2020.

[23] Y. Li, P. Duthon, M. Colomb, and J. Ibanez-Guzman, "What Happens for a ToF LiDAR in Fog?," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 6670–6681, 2021.

[24] C. Goodin, D. Carruth, M. Doude, and C. Hudson, "Predicting the Influence of Rain on LIDAR in ADAS," *Electronics*, vol. 8, no. 1, p. 89, 2019.

[25] S. Hasirlioglu and A. Riener, "A General Approach for Simulating Rain Effects on Sensor Data in Real and Virtual Environments," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 3, pp. 426–438, 2020.

[26] M. Byeon and S. W. Yoon, "Analysis of Automotive Lidar Sensor Model Considering Scattering Effects in Regional Rain Environments," *IEEE Access*, vol. 8, pp. 102669–102679, 2020.

[27] J. P. Espineira, J. Robinson, J. Groenewald, P. H. Chan, and V. Donzella, "Realistic LiDAR With Noise Model for Real-Time Testing of Automated Vehicles in a Virtual Environment," *IEEE Sensors Journal*, vol. 21, no. 8, pp. 9919–9926, 2021.

[28] N. Charron, S. Phillips, and S. L. Waslander, "De-noising of lidar point clouds corrupted by snowfall," in *Proceedings - 2018 15th Conference on Computer and Robot Vision, CRV 2018*, pp. 254–261, 2018.

[29] J. Wu, H. Xu, Y. Tian, R. Pi, and R. Yue, "Vehicle Detection under Adverse Weather from Roadside LiDAR Data," *Sensors*, vol. 20, no. 12, 2020.

[30] M. Ash, M. Ritchie, and K. Chetty, "On the application of digital moving target indication techniques to short-range fmcw radar data," *IEEE Sensors Journal*, vol. 18, 2018.

[31] H. Iqbal, A. Löffler, M. N. Mejdoub, D. Zimmermann, and F. Gruson, "Imaging radar for automated driving functions," *International Journal of Microwave and Wireless Technologies*, vol. 13, 2021.

[32] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 22–35, 2017.

[33] C. H. Kuo, C. C. Lin, and J. S. Sun, "Modified microstrip franklin array antenna for automotive short-range radar application in blind spot information system," *IEEE Antennas and Wireless Propagation Letters*, vol. 16, 2017.

[34] F. R. A. Zakuan, H. Zamzuri, M. A. A. Rahman, W. J. Yahya, N. H. F. Ismail, M. S. Zakariya, K. A. Zulkepli, and M. ZulfaqarAzmi, "Threat assessment algorithm for active blind spot assist system using short range radar sensor," *ARPN Journal of Engineering and Applied Sciences*, vol. 12, 2017.

[35] M. Richards, J. Scheer, and W. Holm, *Principles of modern radar: Basic principles*. Institution of Engineering and Technology, 2010.

[36] H. Wang, B. Wang, B. Liu, X. Meng, and G. Yang, "Pedestrian recognition and tracking using 3D LiDAR for autonomous vehicle," *Robotics and Autonomous Systems*, vol. 88, pp. 71–78, 2017.

[37] M. Parker, "Automotive radar," *Digital Signal Processing 101*, pp. 253–276, 2017.

[38] F. Pfeiffer and E. M. Biebl, "Inductive compensation of high-permittivity coatings on automobile long-range radar radomes," *IEEE Transactions on Microwave Theory and Techniques*, vol. 57, 2009.

[39] Y. Yu, W. Hong, H. Zhang, J. Xu, and Z. H. Jiang, "Optimization and implementation of siw slot array for both medium- and long-range 77 ghz automotive radar application," *IEEE Transactions on Antennas and Propagation*, vol. 66, 2018.

[40] N. V. H and P. V. Venkatesh, "Comparison review on lidar vs camera in autonomous vehicle," *International Research Journal of Engineering and Technology*, 2020.

[41] H. Rashed, E. Mohamed, G. Sistu, V. R. Kumar, C. Eising, A. El-Sallab, and S. Yogamani, "Generalized object detection on fisheye cameras for autonomous driving: Dataset, representations and baseline," *Proceedings - 2021 IEEE Winter Conference on Applications of Computer Vision, WACV 2021*, 2021.

[42] F. E. Sahin, "Long-range, high-resolution camera optical design for assisted and autonomous driving," *Photonics*, vol. 6, 2019.

[43] B. Mohammadian, M. Sarayloo, J. Heil, H. Hong, S. Patil, M. Robertson, T. Tran, V. Krishnan, and H. Sojoudi, "Active prevention of snow accumulation on cameras of autonomous vehicles," *SN Applied Sciences*, vol. 3, 2021.

[44] E. Synge, "XCI. A method of investigating the higher atmosphere," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 9, no. 60, pp. 1014–1020, 1930.

[45] M. A. Tuve, E. A. Johnson, and O. R. Wulf, "A new experimental method for study of the upper atmosphere," *Journal of Geophysical Research*, vol. 40, no. 4, p. 452, 1935.

[46] J. Ring, "The Laser in Astronomy," *New Scientist*, vol. 344, 1963.

[47] A. K. Biswas and W. E. K. Middleton, "Invention of the Meteorological Instruments," *Technology and Culture*, vol. 12, no. 2, 1971.

[48] A. Süss, V. Rochus, M. Rosmeulen, and X. Rottenberg, "Benchmarking time-of-flight based depth measurement techniques," in *Smart Photonic and Optoelectronic Integrated Circuits XVIII*, vol. 9751, p. 975118, 2016.

[49] G. Atanacio-Jiménez, J. J. González-Barbosa, J. B. Hurtado-Ramos, F. J. Ornelas-Rodríguez, H. Jiménez-Hernández, T. García-Ramirez, and R. González-Barbosa, "LIDAR velodyne HDL-64E calibration using pattern planes," *International Journal of Advanced Robotic Systems*, vol. 8, no. 5, pp. 70–82, 2011.

[50] T. Raj, F. H. Hashim, A. B. Huddin, M. F. Ibrahim, and A. Hussain, "A survey on LiDAR scanning mechanisms," 2020.

[51] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems: Review and future perspectives," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, 2014.

[52] P. Wei, L. Cagle, T. Reza, J. Ball, and J. Gafford, "Lidar and camera detection fusion in a real-time industrial multi-sensor collision avoidance system," *Electronics (Switzerland)*, vol. 7, 2018.

[53] S. Hasirlioglu, A. Kamann, I. Doric, and T. Brandmeier, "Test methodology for rain influence on automotive surround sensors," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2242–2247, IEEE, 2016.

[54] T. Ogawa, H. Sakai, Y. Suzuki, K. Takagi, and K. Morikawa, "Pedestrian detection and tracking using

in-vehicle lidar for automotive application," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 734–739, IEEE, 2011.

[55] K. Granstrom, S. Renter, M. Fatemi, and L. Svensson, "Pedestrian tracking using Velodyne data-Stochastic optimization for extended object tracking," in *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 39–46, 2017.

[56] C. Lundquist, K. Granstrom, and U. Orguner, "An Extended Target CPHD Filter and a Gamma Gaussian Inverse Wishart Implementation," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 3, pp. 472–483, 2013.

[57] L. Guo, H. Yihua, L. Zheng, and X. Shilong, "Research on Infulence of Acousto-Optic Frequency Shifter to Micro-Doppler Effect Detection," *Acta Optica Sinica*, vol. 35, no. 2, p. 0212006, 2015.

[58] K. Kidono, T. Miyasaka, A. Watanabe, T. Naito, and J. Miura, "Pedestrian recognition using high-definition lidar," *IEEE Intelligent Vehicles Symposium, Proceedings*, 2011.

[59] S. Hasirlioglu, I. Doric, A. Kamann, and A. Riener, "Reproducible Fog Simulation for Testing Automotive Surround Sensors," in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, vol. 2017-June, pp. 1–7, IEEE, 2017.

[60] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–4, 2011.

[61] H. Balta, J. Velagic, W. Bosschaerts, G. De Cubber, and B. Siciliano, "Fast Statistical Outlier Removal Based Method for Large 3D Point Clouds of Outdoor Environments," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 348–353, 2018.

[62] J. I. Park, J. Park, and K. S. Kim, "Fast and Accurate Desnowing Algorithm for LiDAR Point Clouds," *IEEE Access*, vol. 8, pp. 160202–160212, 2020.

[63] R. Roriz, A. Campos, S. Pinto, and T. Gomes, "DIOR: A Hardware-assisted Weather Denoising Solution for LiDAR Point Clouds," *IEEE Sensors Journal*, vol. 22, no. 2, pp. 1621 – 1628, 2021.

[64] "Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit." [Xilinx, Online; accessed 2021-12-16].

[65] "ROS - Robot Operating System." ROS, [Online; accessed 2021-12-16].

[66] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," *Proceedings - IEEE International Conference on Robotics and Automation*, 2011.

[67] "Qt | Cross-platform software development for embedded & desktop." Qt, [Online; accessed 2021-12-16].

[68] "Openembedded.org." Openembedded, [Online; accessed 2021-12-16].

[69] L. Foundation, "Yocto project quick start," 2015.

[70] "Block Memory Generator." Xilinx, [Online; accessed 2021-12-16].