Hardware-accelerated Data Decoding and Reconstruction for Automotive LiDAR Sensors

Luís Cunha, Ricardo Roriz, Sandro Pinto, and Tiago Gomes

***Abstract*—The automotive industry is facing an unprecedented technological transformation towards fully autonomous vehicles. Optimists predict that, by 2030, cars will be sufficiently reliable, affordable, and common to displace most current human driving tasks. To cope with these trends, autonomous vehicles require reliable perception systems to hear and see all the surroundings, being light detection and ranging (LiDAR) sensors a key instrument for recreating a 3D visualization of the world. However, for a reliable operation, such systems require LiDAR sensors to provide high-resolution 3D representations of the car's vicinity, which results in millions of data points to be processed in real-time. With this article we propose the ALFA-Pi, a data packet decoder and reconstruction system fully deployed on an embedded reconfigurable hardware platform. By resorting to field-programmable gate array (FPGA) technology, ALFA-Pi is able to interface different LiDAR sensors at the same time, while providing custom representation outputs to high-level perception systems. By accelerating the LiDAR interface, the proposed system outperforms current software-only approaches, achieving lower latency in the data acquisition and data decoding tasks while reaching high performance ratios.**

***Index Terms*—Autonomous vehicles, LiDAR, FPGA, Data representation, LiDAR point cloud.**

I. Introduction

Nowadays, just over a decade after the first self-driving car winning the DARPA Challenge [1], the interest in developing fully autonomous vehicles is increasing at a very fast pace. According to optimistic projections, autonomous vehicles will be sufficiently reliable, affordable, and widely spread on our public roads by 2030, replacing many current human driving tasks [2]. Most vehicles today are still manually controlled, and to make them fully autonomous, they still need to go through different levels of driving automation, e.g., the six levels of automation defined by the society of automotive engineers (SAE) [3]. While with SAE-levels 0, 1, and 2, the driver must actively monitor the driving activities, with levels 3, 4, and 5, the automated vehicle should be able to monitor and navigate the environment autonomously. This requires reliable multi-sensor perception systems equipped with RADAR devices, Cameras, and several LiDAR sensors [4]–[6].

A LiDAR sensor works by illuminating a target with a laser pulse and capturing the reflected signal, being the distance to the target obtained by calculating the round-trip time of the travelled light. Since LiDAR sensors work with active illumination, they allow round-the-clock observations, providing accurate measurements up to hundreds of meters even in different weather conditions. Although the adoption of LiDAR sensors in the automotive sector being relatively new, they are assumed as a key technology for the future of autonomous driving due to their ability to recreate the environment through a high-resolution 3D point cloud representation in real-time [7,8]. Despite ongoing challenges, e.g., mutual interference [9,10], and adverse weather [11]–[13], LiDAR sensors can assist perception systems in several tasks such as in detecting obstacles, objects, and vehicles [14]–[16]; pedestrians recognition and tracking [17,18]; ground segmentation for road filtering [19]; among others [20].

For a reliable operation, perception systems require LiDAR sensors to generate millions of data points. For instance, the Velodyne VLS-128, one of the best sensors currently available in the market regarding range and resolution, can provide up to 9.6M points/second. Since a perception system may simultaneously drive several LiDAR sensors within a single vehicle [4,21,22], handling heterogeneous outputs can be quite challenging. LiDAR manufacturers usually provide software layers to interface and collect data from their sensors, enabling an easy and agnostic integration with the processing system. However, it is still required to deal in real-time with the huge amount of data they generate. Some solutions propose to tackle this issue by compressing data after being transferred to a processing unit [23]–[25]. Despite presenting good results in reducing the transmission throughput, the compression tasks can still add significant processing overhead, which may penalize real-time applications that rely on the LiDAR output.

Aiming at exploring hardware-accelerated approaches, this work presents ALFA-Pi, a hardware data packet decoder that can drive LiDAR sensors whose primary interface is an Ethernet port. By decoding sensor's data in hardware, ALFA-Pi can efficiently handle the high throughput output of a LiDAR, even in a multi-sensor configuration, without compromising the performance requirements. Furthermore, the proposed system also includes a data reconstruction module that processes and formats the received data to a customizable representation, supporting different sensor models and data outputs. The main contributions of this article are summarized as follows: (1) a hardware-accelerated data packet decoder to interface any LiDAR sensor with an Ethernet interface; (2) a coordinate system representation converter module that processes the received data, accelerating the calculation and conversion of different coordinate systems; and (3) an extended benchmarking and evaluation of the proposed solution.

Manuscript received xxx; revised xxx. The Associate Editor coordinating the review process was **FistName SecondName**. This work has been supported by FCT - *Fundação para a Ciência e Tecnologia* within the R&D Units Project Scope UIDB/00319/2020, and GRANT 2021.06782.BD.

Luís Cunha, Ricardo Roriz, Sandro Pinto, and Tiago Gomes are with Centro ALGORITMI / LASI, Universidade do Minho, PORTUGAL. Corresponding author: Tiago Gomes (mr.gomes@dei.uminho.pt).

Digital Object Identifier xx.xx/TVT.xx.xx.

II. Background: Automotive LiDAR Sensors

Best-in-class automotive LiDAR sensors provide high resolution and clear-cut images of the vehicle's vicinity in a wide field of view (FoV) and at high frame rates. Since some sensors also come with high range capabilities, they help in enhancing the perception system of the car in the long-distance obstacle detection and avoidance, even at higher speeds. Nonetheless, high throughput sensors require high bandwidth interfaces connected to the processing system, which must deal with several hundreds of Mb/s and handle the decoding and processing of received data in real-time.

A. Sensor interfaces

Regarding their application and operation environment, LiDAR sensors can adopt a variety of communication interfaces, being the Controller Area Network (CAN) bus and Ethernet the most used technologies. CAN is a field bus-level communication standard for automotive that allows computing systems and devices to exchange messages in real-time. It features built-in error detection, great robustness, and extensive flexibility. However, CAN only supports data rates up to 1 Mb/s [26], limiting its utilization to LiDAR sensors with low resolution, small FoV areas, and short-range capabilities. Moreover, and due to vehicles becoming increasingly smart and connected, this traditional interface has to co-exist with other technologies and communication standards that best suite the large amount of distributed computation that require high bandwidth and low latency communications, e.g., Ethernet.

Ethernet is a family of wired networking technologies that has become widely used in on-board and networked automotive systems, such as multimedia/infotainment, sensor interfaces and actuators for ADAS, etc. [27]. A standard Ethernet network can support data rates up to 10 Mb/s. However, recent technologies have increased these transfer rates up to 100 Gb/s (100 Gigabit Ethernet). With such improvement, these interfaces can be adopted by LiDAR sensors with high bandwidth requirements. For interoperability purposes within the vehicle's network, LiDAR sensors output their data using the User Datagram Protocol (UDP) or Transmission Control Protocol (TCP), providing an HTTP-based interface for the configuration and management operations. However, the data packet format is still defined by manufacturers regarding the sensor's model and overall features.

B. LiDAR Output

The output of a LiDAR system is a collection of points, i.e., a point cloud, that together describe the digital 3D representation of the environment around the sensor. Each point in the point cloud can hold accurate information about several measured/calculated parameters such as point coordinates, intensity, timestamps, etc. The coordinate systems used to represent a point's position in the point cloud are usually the cartesian and the spherical systems. In the cartesian system, a point is described by *x, y, z* coordinates, while with the spherical a point within the 3D space is represented by the radius r (distance), the elevation ω (vertical angle), and the azimuth α (horizontal angle), as illustrated by Figure 1.

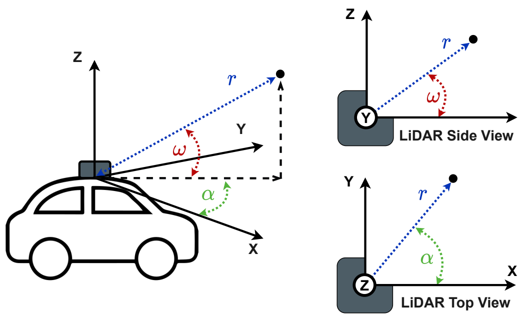

Fig. 1: Spherical coordinate system used by a LiDAR sensor.

Because the cartesian coordinate system is easy to manipulate, it is commonly used in point cloud processing [28]. However, the spherical coordinate system is preferable over the cartesian in LiDAR output data. This is mainly due to the amount of information that can be reduced, i.e., with the spherical coordinate system the value of the elevation angle ω can be easily derived from the data packet according to sensor's design, thus reducing the point cloud data to transmit. For instance, the Velodyne VLP-16 (16 vertically arranged laser channels) transmits for each channel the distance r and the azimuth α for all points. Since the channel is known, the elevation angle is easily obtained. Moreover, and since current LiDAR sensors can provide information of multiple returns for the same emitted light, this data reduction approach creates a significant impact on the sensor's throughput.

When the processing system collects a point cloud, the points can then be rendered as pixels to create accurate 3D models of detected objects. Because the cartesian coordinates are preferable for point cloud processing, sensor manufacturers usually include in their sensor drivers the conversion from spherical data to cartesian coordinates using Equations 1, 2, and 3. For instance, Velodyne sensors come with a Robot Operating System (ROS) package (*velodyne_pointcloud*) that performs in software such point cloud reconstruction. However, and since just one point cloud frame can hold millions of points, this can cause a significant processing overhead.

$$x = r \times \cos\omega \times \sin\alpha \tag{1}$$

$$y = r \times \cos\omega \times \cos\alpha \tag{2}$$

$$z = r \times \sin\omega \tag{3}$$

Moreover, aside from sending the point's coordinates, current LiDAR sensors can also provide extra information concerning target-related characteristics, such as the target's reflectivity and point intensity, which are important to easily detect reflective objects such as traffic signs [29,30] and road boundaries [31]. Using all these data for object detection and avoidance in real-time, requires high bandwidth communication networks and extra computational systems, which usually demand for high hardware resources and performance capabilities. Some solutions already propose hardware-based approaches to accelerate the communication between different distributed computation systems [32], whose technologies can also be specifically applied to interface LiDAR sensors.

III. Related Work

Considering the utilization of hardware-based techniques to interface and handle the high throughput and performance requirements of modern perception systems, and despite hardware-based data decoding approaches being widely applied in other fields, to the best of our knowledge, there are only a few implementations on LiDAR sensors. We believe this is mainly due to automotive LiDAR technology being slightly new and multi-sensor perception systems relying on LiDAR sensors that demand for high bandwidth and low latency communications are still emerging. But as this requirement arises, handling the high LiDAR data throughput will soon become a paramount. Table I summarizes the current LiDAR packet decoding and data reconstruction approaches, somehow related to our solution, that could be found in the literature.

TABLE I: Current LiDAR packet decoding and data reconstruction approaches.

Contribution	Sensor	Operation	Approach	Processing time (ms)
Ning et al. [33] (2017)	VLP-16	Decoding and reconstruction	Software	26.711
Yang et al. [34] (2017)	VLP-16	Decoding and reconstruction	Software	3.855
Fan et al. [35] (2018)	VLP-16	Decoding and reconstruction	Software	-
Okunsky et al. [36] (2018)	VLP-16	Decoding and reconstruction	Software	-
Fan et al. [37] (2019)	VLP-16	Decoding and reconstruction	SoC	0.912
Sun et al. [38] (2019)	SICK LMS111	Packet reading	FPGA	-
Fan et al. [39] (2022)	HDL-64	Decoding and reconstruction	Software	7.678

Regarding software-based approaches, Ning et al. proposed in 2017 an object detection algorithm using a VLP-16 LiDAR sensor, which includes a decoding strategy for retrieving their three-dimensional coordinates to create a point cloud image for horizontal and vertical clustering to obtain the edge of the object [33]. Also, targeting a VLP-16 sensor, Yang et al. explore a method to recreate 3D models of buildings by taking advantage of the 16-angle structure from the Velodyne package to create planes [34]. According to [39], the solutions proposed by Ning et al. [33] and Yang et al. [34] can process one VLP-16 frame in 26.711 ms and 3.855 ms, respectively. Later in 2018 Fan et al. proposed a system architecture to decode, reconstruct and segment raw data from a VLP-16 LiDAR sensor [35]. It supports both single- and dual-mode operation, converting the spherical data to the cartesian coordinates system to perform static object segmentation with a euclidean clustering method. Next, it uses OpenGL for point cloud image construction and visualization. Despite presenting good results in terms of data decoding and object segmentation, further experiments are required to assess the performance evaluation. Okunsky et al. also devised an algorithm with all the necessary steps to decode and reconstruct data from a Velodyne VLP-16 in dual-return mode to the cartesian coordinate system [36]. Although the software results were promising concerning data correctness, further developments and testing with real-time data are required. Finally, in 2022 Fan et al. designed and implemented an algorithm to decode and reconstruct LiDAR data from a HDL-64 sensor [39]. The system was tested with several data packets from the sensor working at 10 Hz in different environments. It can achieve good performance metrics, taking on average around 7.678 ms to process one HDL-64 point cloud frame.

In 2019, and following a hardware-based approach, Fan et al. [37] proposed a system-on-chip (SoC) solution that targets a Velodyne VLP-16 in dual-return mode. The SoC includes a hardware-based parser for data decoding with a lookup-table CORDIC module to convert points from spherical to the cartesian coordinates system. The SoC follows the TSMC 0.18um technology, with a target clock speed operation of 100 MHz. This SoC was further improved to include an enhanced CORDIC design that computes trigonometric functions in seven rotations independently of the number of the input digits. This new version, which uses the rotation input angle to pre-select the desired iteration angle, replaces the previous LUT-based method in a smaller chip area. Acording to [37], the SoC offers good performance metrics in decoding and reconstructing data from a VLP-16 sensor, requiring only 0.012 ms to decode a network packet and 0.912 ms to decode a full frame at the chip's maximum frequency of 100 MHz.

An FPGA-based approach was also proposed by Sun et al. in 2019 [38]. The system resorts to the W5500 chip, a hardwired TCP/IP embedded Ethernet controller used to accelerate the reading of LiDAR data that supports various transport layer protocols along with the 10BaseT/100BaseTX Ethernet standards. The solution was tested with a SICK LMS111 sensor and the communication with the FPGA is done via a serial peripheral interface (SPI) bus. Despite resorting to FPGA, data decoding and reconstruction is not supported in hardware. Although the solution could potentially interface other sensors, it only shows the advantage of not relying entirely on software to handle the TCP/IP network stack of an Ethernet interface when connecting a LiDAR sensor.

IV. Hardware-accelerated LiDAR Interface (ALFA-Pi)

Motivated by the aforementioned challenges, this work proposes a hardware-accelerated data packet decoding and reconstruction system (ALFA-Pi) for automotive LiDAR sensors. This task can be quite challenging since some vehicles already include multi-sensor systems with more than one LiDAR device, which requires different sensor data interfaces and sensor-specific packet decoding for point cloud reconstruction.

A. System Architecture

Figure 2 depicts the ALFA-Pi architecture, which following a hardware-software co-design approach, enables the high-speed reading, decoding, and reconstruction of LiDAR data. ALFA-Pi is built upon the ZCU104 Evaluation Kit, which features a Zynq UltraScale+ MPSoC, enabling the rapid design of embedded applications with support to video codecs and standard peripherals and interfaces. The MPSoC combines two distinct systems, the processing system (PS) and the programmable logic (PL), which includes a quad-core Arm

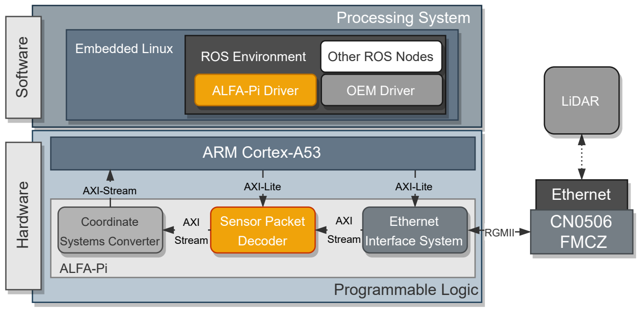

Fig. 2: ALFA-Pi system architecture.

Cortex-A53 application processor, a dual-core Cortex-R5 real-time processor, a Mali-400 MP2 graphics processing unit, 4KP60 capable H.264/H.265 video codecs, configurable field-programmable gate array (FPGA) technology, and 2 GB of DDR4 memory. To interface different LiDAR sensors, the ALFA-Pi includes the EVAL-CN0506-FMCZ expansion board. This board is a dual-channel, low latency, low power Ethernet PHY card that supports 10 Mbps, 100 Mbps, and 1000 Mbps speeds for industrial Ethernet applications. The EVAL-CN0506-FMCZ board directly connects to the PL, enabling the deployment in FPGA of hardware peripherals to collect and process LiDAR data.

Regarding the PL system, it features three main hardware blocks: (1) the ***Ethernet Interface System***; the (2) ***Sensor Packet Decoder***; and (3) the ***Coordinate Systems Converter (CSC)***, used to accelerate the conversion between different coordinate systems. Combined, these blocks are responsible for the reading, decoding, and reconstruction of LiDAR data, and together they form the ALFA-Pi hardware system. All modules are connected through the advanced extensible interface (AXI)-4 bus, which is also used for high-performance memory accesses, high-speed point cloud data streaming, and the control/configuration operations. From these hardware blocks, both the ***CSC*** and the ***Ethernet Interface System*** are only deployed once, not requiring further modifications when the support for a new sensor is added. However, adding support for a new sensor requires modifications to the ***Sensor Packet Decoder***, regarding the data decoding and reconstruction to the ALFA-Pi format.

On the PS side, the ALFA-Pi features a ***ROS environment*** on top of an embedded Linux distribution, allowing for real-time configuration and control of the hardware functionalities through the developed ALFA-Pi ROS Package (ALFA-Pi driver). Therefore, high-level applications can easily access data through standard ROS interfaces, as commonly provided by manufacturers and sensors.

B. Ethernet Interface System

The ***Ethernet Interface System*** allows collecting data directly from the Ethernet EVAL-CN0506-FMCZ board interface, accelerating the reading task without the need of the operating system (OS) intervention, and making it directly available to the ***Sensor Packet Decoder*** block. This system resorts to several Xilinx intellectual property (IP) Cores, e.g., ADIN1300 to implement the physical (PHY) layer, and the AXI 1G/2.5G Ethernet Subsystem for the media access controller (MAC) layer. The MAC layer controls the data-link layer of the open systems interconnection (OSI) model, thus, it can be easily intercepted in hardware to capture IEEE 802.3-compliant data frames. The remaining upper layers, e.g., internet protocol version 4 (IPv4) and transport layers (e.g., UDP), are still controlled by the network stack manager provided by the embedded Linux OS. Although these could be deployed also in hardware, they would not improve the packet decoding and reconstruction process. After performing all MAC-related operations, e.g., flow control, and data frame integrity, received data is sent to the ***Sensor Packet Decoder*** through an AXI4-Stream interface for the filtering and decoding tasks, according to the sensor(s) connected.

C. Sensor Packet Decoder

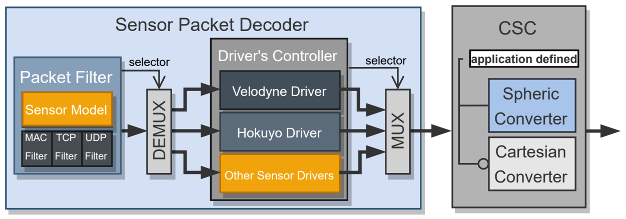

Fig. 3: Sensor Packet Decoder block diagram.

The ***Sensor Packet Decoder***, depicted in Figure 3, is responsible for the high-speed processing and decoding of received data. It is composed of two main blocks: (1) a packet filter for filtering the received data frame; and (2) sensor-specific interface blocks that implement all functionalities as provided by respective original equipment manufacturer (OEM) sensor drivers. The packet filter supports different filtering features of several protocols, e.g., MAC and IPv4 addresses, and TCP and UDP headers/ports. Also, it can verify the type and settings (e.g., single- or dual-mode return) of each LiDAR sensor connected and configured in the ALFA-Pi interface. When a received packet belongs to a supported sensor currently active in the system, the ***Packet Filter*** module forwards it to the corresponding sensor's driver controller. Otherwise, the packet is discarded. All registers and configuration parameters of this module can be set in run-time from the ALFA-Pi driver.

Each sensor interface module is a sensor-specific hardware block that performs in hardware the same functionalities supported by the software drivers provided by manufacturers, thus adding a new sensor, requires the deployment of sensor-defined features with fine-tuning configurations on subsequent sub-modules. Nonetheless, these blocks do not need further changes once a sensor model/family is added. Since ALFA-Pi can support different sensors connected and working at the same time, the ***Sensor Packet Decoder*** features a multiplexing system that combines all sensors data into a single point cloud stream following the ALFA-Pi's point cloud representation. Regardless the sensor data that is being decoded, the output of this module to the CSC follows the ALFA-Pi Data Format (further explained in Section IV-D), which by default contains data points in the spherical representation. For demonstration purposes, this article only focuses on the support for Velodyne and Hokuyo LiDAR sensors, further evaluated in Section V.

Velodyne Sensors Support

Since Velodyne sensors adopt a generic packet structure in their point cloud output, the ALFA-Pi platform already supports a wide range of Velodyne LiDAR products. As depicted in Figure 4, without considering the UDP header (42 bytes), the data block structure is composed of 12 data blocks (1200 bytes), a timestamp field (4 bytes), and sensor-related info (2 bytes) that contain the sensor's model and current operation mode. Each data block N contains 32 data fields to store the point's information (distance and reflectivity) of each available laser channel for the azimuth N angle. The flag 0xFF(XX) identifies the firing sequence within the data block. For instance, if a sensor with 16 or 32 laser channels is being used, e.g., the VLP-16 or HDL-32E sensors, one data block can hold information of one or two firing sequences. By its turn, if the HDL-64E sensor is connected, two data blocks are required to store the 64 channels of the firing sequence, being the first and second data blocks identified by the flags 0xFFEE and 0xFFDD, respectively. Additionally, when the sensor is being used in dual return mode, twice as many packets are returned since more data blocks are needed for sending the different returns from the same laser firing, e.g., for the HDL-32E the data block N and the data block N+1 would contain the information for points in the same azimuth N.

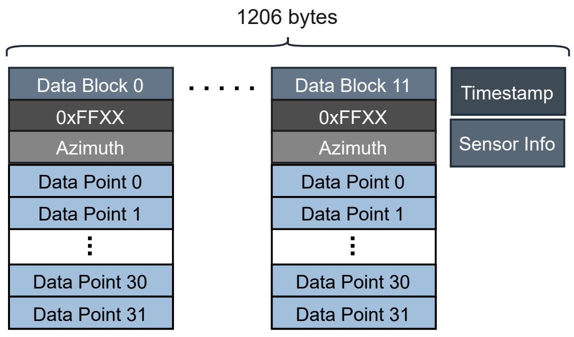

Fig. 4: Velodyne packet format.

The ***Velodyne Sensor Interface*** block can detect the sensor model and its operation mode directly from the received packet. Thus, it can decode data from different Velodyne sensors simultaneously connected to the ALFA-Pi. However, this is only possible after a full packet is received. Internally, this hardware module features a message controller and a data decoder system. The message controller takes advantage of the packet format to extract and place the distance and intensity values into different memory blocks. By its turn, the data decoder system is responsible for decoding the packet data and organizing it into the ALFA-Pi data format.

Hokuyo Sensors Support

To show the modularity and heterogeneous features of the ALFA-Pi, the platform also provides support to sensors with different frame formats. For instance, Hokuyo's 2D sensors use the SCIP2.0 protocol, which is currently adopted by different sensors with distinct operation modes. This protocol, whose frame format is depicted in Figure 5, is based on a command-response principle, where the host processing system sends a command/request and the sensor responds with the requested data. The sensor data is structured in blocks of 64 bytes, containing the data points stored with character encoding and a check code (CC) character, which can be used for block error detection purposes. Because each data point only contains the distance and/or the intensity information, the azimuth values can be inferred from the data point's position inside the message array.

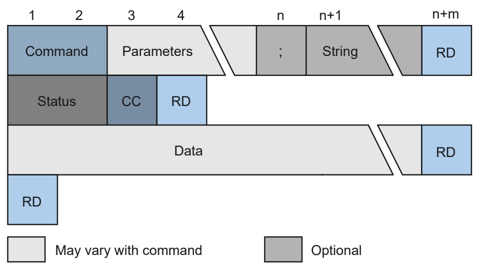

Fig. 5: Hokuyo packet format.

Unlike the frame format used in Velodyne sensors, where data packets are independent of each other, the Hokuyo's format requires several TCP packets to be received until an entire message can be successfully decoded. For this reason, the Hokuyo's Sensor Interface block features several data buffers to store multiple TCP packets from a single sensor until a complete scan message is received. Storing all packets for a specific sensor within the same hardware-based data buffer can lead to higher memory utilization, however, it enables an easier control and a faster decoding operation. The Hokuyo hardware interface also contains a message controller, which is responsible for controlling all data buffers and routing received packets regarding the corresponding sensor's ID. When a full sensor's message is received, the decoding modules are triggered and the decoding operation starts to fetch data from the correspondent data buffers. The output of this module is a data frame that follows the ALFA-Pi's data format.

D. ALFA-Pi Data Format

One of the biggest advantages of ALFA-Pi's is the support for distinct sensors that can be used in different automotive LiDAR applications. For instance, Velodyne sensors are 3D sensors commonly used in long-range driving assistance tasks while the Hokuyo 2D sensors can be used in short-range jobs such as parking and blind spot assistance. Nonetheless, the ***Sensor Packet Decoder*** module is still able to interface both sensors at the same time, providing a custom data format that contains all sensor's data regardless of their original output. Such abstraction layer eases the integration of the ALFA-Pi framework with other sensors, while seamlessly connecting high-level applications that integrate the perception system of the car. The proposed format is able to hold information for both 2D and 3D sensors, and support sensors with multiple return capabilities.

By default, ALFA-Pi data format includes the spherical coordinates system, as provided by most of the sensors available in the market, generating the output depicted in Figure 6. The data structure starts with the ***Flags*** field, which indicates when a new packet frame starts, followed by the ***Sensor ID*** field (6 bits) that identifies the sensor in the system. The next fields contain the ***Azimuth*** (16-bit) and the ***Vertical Angle*** (16-bit), followed by the ***D0*** and ***D1*** dual return support (when present), both requiring 20 bits each. The two distance values ***D0*** and ***D1***, contain the point's distance obtained by the sensor (in millimeters) for each return. Finally, the ***Extra 0*** and ***Extra 1*** fields, 8 bits each, can hold extra information for other sensor-specific features, e.g., reflectivity and packet time stamps.

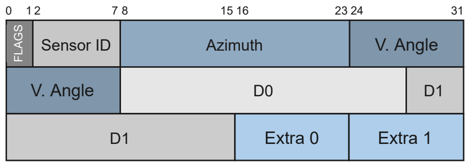

Fig. 6: ALFA-Pi spherical system representation (96 bit).

E. Coordinate Systems Converter (CSC) module

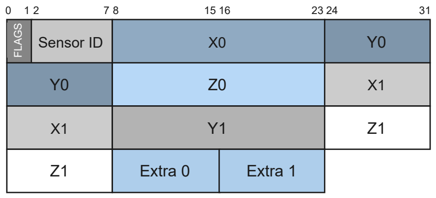

Fig. 7: ALFA-Pi cartesian system representation (120 bit).

The ***CSC*** module is responsible for handling the representation of the received sensor data, supporting the conversion between two coordinate systems. Customized on-the-fly, it can convert spherical coordinates to the cartesian system, and vice-versa. These conversions are usually performed by sensor's drivers, which are traditionally supported by ROS-based software packages. Since trigonometric calculations (Equations 1, 2, and 3) require time consuming processing, ALFA-Pi deploys them in hardware. However, calculating the *sine* and *cosine* values with a hardware architecture that does not provide specialized calculation units comes at some hardware costs. There are two well-known techniques that can be adopted to perform such calculations, such as the CORDIC algorithm or a strategy based on look-up tables. Despite being more resource-consuming, the ***CSC*** module deploys the look-up tables approach, which allows for better performance results.

The ALFA-Pi's look-up table implementation resorts to a block RAM (BRAM) memory and uses a table/array of already computed *sine* and *cosine* values, turning the run-time computation into a simple indexing operation. This allows for a significant processing time reduction since retrieving the value from memory is faster than its recurring computation. The look-up tables were populated with the *sine* and *cosine* values, calculated with the C math library with a resolution of 0.01º for angles between 0º and 90º, and store five decimal digits for each calculated value. When high-level applications require the point cloud in the cartesian representation, the output of the ***Sensor Packet Decoder*** is intercepted by the ***CSC*** module, which converts the spherical coordinates data to the format depicted by Figure 7. This frame format is similar to the spherical system representation, however it contains the x, y, z coordinates instead of the azimuth and elevation angles, and requires a packet size of 120 bits instead of 96 bits. Regardless of the frame format, the point cloud data are written by a hardware direct memory access (DMA) module into the DDR4 memory available in the ZCU 104 platform.

V. Evaluation

The evaluation of the ALFA-Pi aims at comparing a software-based LiDAR setup (native ROS driver; software-based Ethernet port; Linux network stack; and no acceleration support) with the proposed hardware-based solution (hardware Ethernet port; tweaked network stack/interface; ALFA-Pi hardware modules). It also includes a performance comparison with state-of-the-art solutions, the FPGA resources required to deploy the ALFA-Pi, and the power consumption of the ALFA-Pi hardware modules. All software runs on top of an embedded Linux (4.19.0-xilinx-v2019.2) with a ROS environment (Ros1 melodic distribution). The hardware platform combines the ZCU104 Evaluation Kit and the EVAL-CN0506-FMCZ Ethernet board, with the CPU running at a clock speed of 1.2 GHz and the FPGA fabric running at 100 MHz.

A. Network Interface

TABLE II: Network performance results from iPerf.

	Test	Native Ethernet	Hardware Ethernet	ALFA-Pi
TCP	Bandwidth (Mbits/s)	941.243	941.904	941.302
	Retries	0	0	0
UDP	Bandwidth (Mbits/s)	955.937	955.929	955.971
	Packet Loss (%)	0.002	0.001	0.002
	Jitter (ms)	0.018	0.018	0.018

For testing the impact of adding the hardware Ethernet port, we ran the iPerf3 tool on top of the embedded Linux. We used the iPerf client-server functionality for both UDP and TCP data streams, testing their most relevant metrics in three possible configurations: (1) native Ethernet port of the ZCU104 Evaluation Kit; (2) hardware Ethernet interface from the EVAL-CN0506-FMC card; and (3) hardware Ethernet interface EVAL-CN0506-FMC with ALFA-Pi accelerators. In all configurations the network stack is managed by the Linux network manager. The embedded platform was directly connected to a desktop Linux host with a 1 Gbit/s enabled Ethernet interface. Table II summarizes the gathered results. With the TCP, the maximum bandwidth rate is around 941 Mbits/s in all three configurations, while for the UDP test the maximum bandwidth is nearly 955 Mbits/s, also in all setups. The TCP number of retries is always zero, and in the UDP tests the lost packet ratio is as low as 0.002%. The measured jitter is always 0.018 milliseconds (ms). From the obtained results we can conclude that adding the hardware-based network interface and the ALFA-Pi accelerators does not improve nor affect the normal behaviour of the network communication.

B. Performance Evaluation

For this evaluation we started to benchmark a native software-based solution with the hardware platform (without ALFA-Pi) connected to a Velodyne and a Hokuyo sensor (Table III, column ***Native ROS driver***). Next, we repeat the same evaluation steps with the ALFA-Pi hardware modules, including the ***CSC*** for fast point cloud reconstruction (column ***ALFA-Pi driver***). The performance gains between the software and hardware version are included in column $Gain_1$. For a fair comparison, in each sensor evaluation we have used the same point cloud data (i.e., raw Ethernet packets previously captured from sensors) on both native and ALFA-Pi solutions.

One big advantage of the ALFA-Pi is the ability to provide the received and processed point cloud frames directly to other hardware accelerators. This way, instead of using the software stack to collect data and perform software-based data processing, other accelerators can benefit from the hardware capabilities of ALFA-Pi by directly collecting data from the ***CSC*** module. These results are present in column ***ALFA-Pi hw***) and the respective performance gains between the software-only and the full hardware approach can be found in column $Gain_2$.

TABLE III: Comparison of the native implementation vs. ALFA-Pi ROS driver and ALFA-Pi hardware.

Sensor	Point Cloud Size	Native ROS driver (ms)	ALFA-Pi driver (ms)	Gain_1	ALFA-Pi hw (ms)	Gain_2
UST-10LX (Distance)	1081	2.642	0.044	60.73x	0.032	81.55x
UST-10LX (Distance+Intensity)	1081	2.647	0.076	34.87x	0.064	40.86x
VLP-16	29184	8.643	0.929	9.3x	0.613	14.16x
HDL-32	69504	21.586	2.212	9.76x	1.464	14.78x
HDL-64	133632	40.380	4.254	9.5x	2.821	14.32x
VLS-128	240384	72.113	7.651	9.42x	5.064	14.2x

Discussion: Native ROS driver vs. ALFA-Pi

After the point cloud data being received by the Ethernet interface, the native ROS sensor's driver reads it from the Linux network stack layer, reconstructs in software the point cloud with the cartesian coordinate system, and makes it available to other applications on a new ROS topic. By its turn, the ALFA-Pi hardware blocks read the sensor data directly from the Ethernet interface, performs the packet decoding according to the sensor's model, and performs the point cloud data reconstruction with the ***CSC*** hardware block. Finally, the ALFA-Pi's software driver in the PS reads data directly from the DDR4 memory.

Regarding the Hokuyo sensor, with its native ROS driver, it is necessary around 2.642 ms to process one point cloud of 1081 points/frame when the sensor outputs only the point's coordinates, and 2.647 ms when the intensity values are added to the packet. For the Velodyne sensors, the ROS driver take around 8.64 ms for the VLP-16 (29184 points/frame), 21.58 ms for the HDL-32 (69504 points/frame), 40.38 ms for the HDL-64 (133632 points/frame), and 72.11 ms for the VLS-128 (240384 points/frame). Despite the big difference between the Hokuyo and the VLP-16 on the point cloud sizes, the processing time is only 3.27x higher for the VLP-16. This can be explained by the packet decoding scheme used by Hokuyo, which requires several data packets to be received and stored in memory until a point cloud frame could be decoded.

When resorting to the ALFA-Pi hardware packet decoder and the ***CSC*** module, the proposed system can achieve a performance gain (column Gain_1) up to nearly 60x for the Hokuyo sensor when its output is only the point's coordinates (horizontal angle and distance), and 34x when the intensity values are added. This improvement is around 9.3x for the VLP-16, 9.76x for the HDL-32, 9.5x for the HDL-64, and 9.42x for the VLS-128. The performance gains of the Hokuyo sensor are mainly related to the optimizations performed in the packet decoder block in terms of point cloud reconstruction, memory utilization, and hardware resources. Thus, a reduced point cloud size will always decode faster. However, adding the intensity values to the Hokuyo's point cloud reduces the performance gain almost by the half, which is explained by the point cloud size requiring nearly twice the packets to include such data. For the Velodyne sensors, since they use the same packet format, the performance gains are almost identical, being the frame processing time only related to the point cloud size, rather than the packet decoding process.

When directly connecting the output of ALFA-Pi to other hardware accelerators, the performance gains in processing a full point cloud frame are further increased (column Gain_2). For the Velodyne sensors these gains are around 14x, while for the Hokuyo the performance gain can be improved to 40.86x when including the Itensity values and 81.55x when only operating in the Distance mode. In general, the performance gains obtained in the ALFA-Pi hardware-only evaluation can be explained by the reduction in the memory access time and the overhead induced by the upper software layers.

ALFA-Pi vs. state-of-the-art solutions

The comparison between the ALFA-Pi driver and current state-of-the-art solutions can be found in Table IV, which only includes contributions that provide the information about the processing time required to handle one point cloud frame. Regarding the VLP-16, ALFA-Pi can achieve a frame processing time of 0.929 ms, outperforming the contributions

TABLE IV: Comparison with state-of-the-art solutions.

Contribution	Sensor	Point Cloud Size	Processing time (ms)	Approach
Ning et al. [33] (2017)	VLP-16	29184	26.711	Software
Yang et al. [34] (2017)	VLP-16	29184	3.855	Software
Fan et al. [37] (2019)	VLP-16	29184	0.912	SoC
Fan et al. [39] (2022)	HDL-64	133632	7.678	Software
ALFA-Pi (2022)	UST-10LX (Distance)	1081	0.044	FPGA
	UST-10LX (Distance+Intensity)	1081	0.076	FPGA
	VLP-16	29184	0.929	FPGA
	HDL-32	69504	2.212	FPGA
	HDL-64	133632	4.254	FPGA
	VLS-128	240384	7.651	FPGA

from Ning et al. [33], 26.711 ms, and Yang et al. [34], 3.855 ms. In Fan et al. [37], the processing time for the VLP-16 is 0.912 ms, obtained in a hardware-only configuration, i.e., no software drivers. Thus, for a fair comparison with this SoC-based implementation, we must also use the hardware-only ALFA-Pi, which corresponds to 0.613 ms. Nonetheless, ALFA-Pi driver can still achieve a processing time of 0.929 ms, which is only 1.87% worse than the SoC. For the HDL-64, the contribution from Fan et al. [39] processes one point cloud frame in 7.678 ms, while the ALFA-Pi requires only 4.254 ms.

C. Hardware Resources

All features of the ALFA-Pi come at the cost of FPGA hardware resources, which may vary with the number of sensors connected to the system. Table V summarizes all the required FPGA blocks (from the available in the Zynq UltraScale+ MPSoC) in terms of look-up tables (LUTs), look-up table random access memories (LUTRAMs), Flip-Flops (FFs), and BRAMs, for the deployment of the ***Velodyne Interface*** and ***Hokuyo Interface*** blocks, the ***Packet Filter*** module, the ***CSC*** block, and the ***Hardware Ethernet*** interface. Each module was evaluated independently, wherefore different configurations and customizations of an ALFA-Pi setup will achieve different resources consumption. We also estimate the number of sensors (1 to 16) that could be simultaneously interfaced by ALFA-Pi in a multi-sensor configuration.

TABLE V: Hardware resources utilization.

	Sensors	LUTs (230400)	LUTRAMs (101760)	FFs (460800)	BRAMs (312)
Hardware Ethernet	-	4683 (2.03%)	556 (0.546%)	9213 (2%)	5.5 (1.76%)
Packet Filter	1	107 (0.046%)	0 (0%)	262 (0.057%)	0 (0%)
	2	218 (0.095%)	0 (0%)	334 (0.072%)	0 (0%)
	4	264 (0.12%)	0 (0%)	427 (0.093%)	0 (0%)
	8	358 (0.16%)	0 (0%)	612 (0.13%)	0 (0%)
	16	567 (0.25%))	0 (0%)	981 (0.21%)	0 (0%)
Velodyne Interface	1	419 (0.18%)	16 (0.016%)	360 (0.078%)	4 (0.64%)
	16	419 (0.18%)	16 (0.016%)	360 (0.078%)	4 (0.64%)
Hokuyo Interface	1	196 (0.085%)	16 (0.016%)	421 (0.091%)	2 (0.64%)
	2	316 (0.14%)	16 (0.016%)	514 (0.11%)	5 (1.6%)
	4	490 (0.21%)	16 (0.016%)	700 (0.15%)	10 (3.2%)
	8	865 (0.38%)	16 (0.016%)	1072 (0.23%)	20 (6.41%)
	16	1586 (0.68%)	16 (0.016%)	1816 (0.39%)	40 (12.82%)
CSC	-	1350 (0.58%)	0 (0%)	57 (0.012%)	17 (5.45%)

The ***Ethernet Interface System*** was deployed straight out of the box and does not feature any customization parameters. Each independent Ethernet port requires 4683 LUTs, 556 LUTRAMs, 9213 FFs, and 5.5 BRAMs. Regarding the ***Sensor Packet Decoder*** block, which includes the ***Packet Filter*** and at least one sensor interface, it can present different configurations regarding the number of sensors connected to the system, e.g., for 4 Velodyne sensors, the ***Packet Filter*** must include 4 filtering blocks. Connecting 16 sensors to the system keeps the resource requirements as low as 567 LUTs, 981 FFs, and zero LUTRAMs and BRAMs units. These resources are not affected by the model/type of the sensor connected.

The ***Velodyne Sensor Interface*** provides great scalability features due to its generic packet format implementation. Therefore, deploying 1 or 16 sensors provides the same resources utilization, i.e., 419 LUTs, 16 LUTRAMs, 360 FFs, and 4 BRAM modules, since each received packet does not require any storage steps for packet decoding. The ***Hokuyo Sensor Interface*** highly depends on the number of sensors connected to ALFA-Pi. This is mainly due to the storage resources required in the packet reconstruction process. Therefore, the most critical resource is the required number of BRAMs (the scarcest resource) used to accommodate the received Hokuyo packets, which exponentially increases when the number of connected sensors also increases. Connecting 16 Hokuyo sensors would require 1586 LUTs, 16 LUTRAMs (this value never changes), 1816 FFs, and 40 BRAMs.

The ***CSC*** module is responsible to perform the fast conversion between the two coordinate systems used in ALFA-Pi. Since it uses LUTs to immediately find the corresponding x, y, z values of a point inside multiple tables stored in BRAMs, it requires 1350 LUTs and 17 BRAMs, being the LUTRAMs, and FFs as low as 0 and 57, respectively.

D. ALFA-Pi Power Consumption

To assess the power consumption of the supported hardware interfaces running simultaneously with different sensor configurations, we used the Power Analysis tools from the Vivado Design Suite. The tool was run in vectorless mode with the default settings, the platform constrains for the Zynq UltraScale+ MPSoC present in the ZCU104 Evaluation Kit, different sensor configurations (support for 1 and 16 sensors), and with power optimizations disabled.

Table VI summarizes the dynamic power consumption, which is determined by the switching activity of clocks and datapaths and the static power consumption, which represents the minimum power consumption required to operate the hardware blocks. Additionally, and since the frame processing time is known, we can calculate the required energy to process a point cloud frame for the supported sensors.

TABLE VI: ALFA-PI power consumption.

	Sensors	Dynamic Power (W)	Static Power (W)	Total (W)	Sensor Model	Energy per frame (mJ)
Velodyne Interface	1 16	0.226 0.229	0.594 0.594	0.820 0.823	**VLP-16**	0.765
					HDL-32	1.819
					HDL-64	3.498
					VLS-128	6.296
Hokuyo Interface	1 16	0.224 0.259	0.595 0.595	0.820 0.854	**UST-10LX (D)**	0.037
					UST-10LX (D+I)	0.065

Considering the total power, the ***Velodyne Interface*** dissipates 0.820 W for a configuration with 1 sensor, and 0.823 W when handling 16 sensors. This small change is related to the hardware required to support 1 or 16 sensors, which due to the Velodyne's generic packet format, allows for a good hardware re-utilization. Processing one VLP-16 frame consumes 0.765 mJ while for the HDL-32 the dissipated energy is 1.819 mJ. The HDL-64 and the VLS-128 dissipate, respectively, 3.498 mJ and 6.296 mJ. These values are directly dependent on the point cloud's frame size, which requires more processing time when more point cloud data is genereted by the sensor.

Regarding the ***Hokuyo Interface***, interfacing 1 sensor dissipates also 0.820 W, while supporting 16 sensors consumes around 0.854 W. These differences are directly justified by the increased hardware resources, e.g., buffers for accommodating the received frames, that need to be added to the module when more sensors are being used. It consumes around 0.037 mJ to process one frame in the basic sensor's configuration, and 0.065 mJ when the intensity values are also included.

VI. Conclusion

Triggered by the automotive sector, the LiDAR market is growing at a breathtaking pace. Current perception systems based on multi-sensor approaches can provide more accurate readings of the surroundings, but require more processing capabilities to handle the high volume of data produced by several sensors working at the same time. This article proposes a hardware-based solution for packet decoding and data reconstruction for automotive LiDAR sensors. Contrarily to other state-of-the-art solutions, it can support different sensors connected via an Ethernet interface, achieving good performance gains when compared with software-only state-of-the-art contributions. Despite only presenting results for Hokuyo and Velodyne, we are currently working on adding more sensors from different manufacturers to provide support for a multitude of configurations and sensor scenarios.

References

[1] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge." *J. Field Robot.*, vol. 23, pp. 661–692, 2006.

[2] T. Litman, *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada, 2021.

[3] Society of Automotive Engineers (SAE), "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (Surface Vehicle Recommended Practice: Superseding J3016 Jun 2018)." SAE International, Apr. 2021.

[4] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[5] 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.

[6] 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.

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

[8] R. Roriz, J. Cabral, and T. Gomes, "Automotive LiDAR Technology: A Survey," *IEEE Trans. on Intell. Transp. Syst*, pp. 1–16, 2021.

[9] G. Kim, J. Eom, and Y. Park, "An Experiment of Mutual Interference between Automotive LIDAR Scanners," in *2015 12th International Conf. on Inf. Technol. - New Generations*, 2015, pp. 680–685.

[10] I.-P. Hwang, S.-J. Yun, and C.-H. Lee, "Mutual interferences in frequency-modulated continuous-wave (FMCW) LiDARs," *Optik*, vol. 220, p. 165109, 2020.

[11] A. M. Wallace, A. Halimi, and G. S. Buller, "Full Waveform LiDAR for Adverse Weather Conditions," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7064–7077, 2020.

[12] N. Charron, S. Phillips, and S. L. Waslander, "De-noising of Lidar Point Clouds Corrupted by Snowfall," in *2018 15th Conference on Computer and Robot Vision (CRV)*, 2018, pp. 254–261.

[13] 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, 2022.

[14] 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 Trans. on Intell. Transp. Syst*, vol. 20, no. 10, pp. 3782–3795, 2019.

[15] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud," *2019 IEEE/CVF Conf. on Comput. Vision and Pattern Recognition (CVPR)*, pp. 770–779, 2019.

[16] 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.

[17] 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.

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

[19] 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.

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

[21] M. Sualeh and G.-W. Kim, "Dynamic Multi-LiDAR Based Multiple Object Detection and Tracking," *Sensors*, vol. 19, no. 6, 2019.

[22] 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.

[23] I. Maksymova, C. Steger, and N. Druml, "Review of LiDAR Sensor Data Acquisition and Compression for Automotive Applications," *Proceedings 2018*, vol. 2, p. 852, 12 2018.

[24] P. Caillet and Y. Dupuis, "Efficient LiDAR data compression for embedded V2I or V2V data handling," *ArXiv*, vol. abs/1904.05649, pp. 1–6, 2019.

[25] M. M. Abdelwahab, W. S. El-Deeb, and A. A. A. Youssif, "LIDAR Data Compression Challenges and Difficulties," in *2019 5th International Conf. on Frontiers of Signal Processing (ICFSP)*, 2019, pp. 111–116.

[26] International Organization for Standardization, "Road vehicles — Controller area network (CAN) — Part 1 : Data link layer and physical signalling," 2015. [Online]. Available: https://www.iso.org/standard/63648.html

[27] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2019.

[28] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE Int. Conf. on Robotics and Automation*, 2011, pp. 1–4.

[29] S. Weng, J. Li, Y. Chen, and C. Wang, "Road traffic sign detection and classification from mobile LiDAR point clouds," *2nd ISPRS Int. Conf. on Computer Vision in Remote Sensing (CVRS 2015)*, vol. 9901, no. Cvrs 2015, p. 99010A, 2016.

[30] S. Gargoum, K. El-Basyouny, J. Sabbagh, and K. Froese, "Automated Highway Sign Extraction Using Lidar Data," *Transportation Research Record*, vol. 2643, pp. 1–8, 2017.

[31] B. Yang, Z. Wei, Q. Li, and J. Li, "Automated extraction of street-scene objects from mobile lidar point clouds," *International Journal of Remote Sensing*, vol. 33, no. 18, pp. 5839–5861, 2012.

[32] S. Shreejith, P. Mundhenk, A. Ettner, S. A. Fahmy, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "VEGa: A High Performance Vehicular Ethernet Gateway on Hybrid FPGA," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1790–1803, 2017.

[33] H.-I. Ning and Y.-C. Fan, "LiDAR information for objects classified technology in static environment," in *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2017, pp. 125–126.

[34] S.-C. Yang and Y.-C. Fan, "3d building scene reconstruction based on 3d lidar point cloud," in *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2017, pp. 127–128.

[35] Y.-C. Fan, L.-J. Zheng, and Y.-C. Liu, "3D Environment Measurement and Reconstruction Based on LiDAR," in *2018 IEEE Int. Instrumentation and Measurement Technology Conference (I2MTC)*, 2018, pp. 1–4.

[36] M. V. Okunsky and N. V. Nesterova, "Velodyne LIDAR method for sensor data decoding," *IOP Conference Series: Materials Science and Engineering*, vol. 516, p. 012018, apr 2019.

[37] Y.-C. Fan, Y.-C. Liu, and C.-A. Chu, "Efficient CORDIC Iteration Design of LiDAR Sensors' Point-Cloud Map Reconstruction Technology," *Sensors*, vol. 19, no. 24, 2019.

[38] T. Sun, Y. Liu, and Y. Wang, "Design and implementation of a high-speed lidar data reading system based on FPGA," *2019 IEEE Int. Conf. on Real-Time Computing and Robotics, RCAR 2019*, pp. 322–327, 2019.

[39] Y.-C. Fan and S.-B. Wang, "Three-Dimensional LiDAR Decoder Design for Autonomous Vehicles in Smart Cities," *Inf.*, vol. 13, no. 1, 2022.

Luís Cunha has received his master's degree in Industrial Electronics and Computers Engineering from the University of Minho, Braga, Portugal. Currently, he is an active research fellow at the Embedded Systems Research Group within the ALGORITMI Research Center. His interests include hardware acceleration systems, embedded systems design, and automotive technology. Contact him at a82307@alunos.uminho.pt.

Ricardo Roriz has received his master's degree in Industrial Electronics and Computers Engineering from the University of Minho, Portugal. Currently, he is pursuing a Ph.D. degree in sensors and instrumentation systems for automotive applications. He is an active research fellow at the Embedded Systems Research Group within the ALGORITMI Research Center. His research interests include embedded systems design and ADAS, with special focus on FPGA hardware acceleration. Contact him at id8677@alunos.uminho.pt.

Dr. Sandro Pinto is an Associate Research Professor at the University of Minho, Portugal. He holds a Ph.D. in Electronics and Computer Engineering. Sandro has a deep academic background and several years of industry collaboration focusing on operating systems, virtualization, and security for embedded, CPS, and IoT systems. He has published 70+ scientific papers in top-tier conferences/journals and is a skilled presenter with speaking experience in several academic and industrial conferences. Contact him at sandro.pinto@dei.uminho.pt.

Dr. Tiago Gomes has received the master's degree in telecommunications engineering and Ph.D. degree in electronics and computers engineering from the University of Minho, Braga, Portugal. He is a Research Scientist and Invited Professor with the University of Minho. His current research interests include embedded hardware acceleration for autonomous perception systems based on LiDAR sensors, and hardware/software co-design for resource constrained Internet of Things devices. Contact him at mr.gomes@dei.uminho.pt.