UCLouvain

École polytechnique de Louvain

Camera, radar and lidar sensor synchronization and dataset creation for traffic monitoring

Author: **Edwin Desjardin**
Supervisors: **Christophe Craeye, Benoît Macq**
Readers: **Jérôme Louveaux, Gauthier Rotsart de Hertaing**
Academic year 2023–2024
Master [120] in Electrical Engineering

Acknowledgements

These pages are the culmination of an enriching five years journey. It included wonderful meetings and many opportunities for learning. This work concludes a chapter of my life, from which I will keep beautiful memories. This thesis taught me to be rigorous and introduced me to the field device synchronization and sensors data processing.

I would like to express my deepest appreciation to Pr. Christophe Craeye and Pr. Benoît Macq for their support and time investment in this thesis. Their experience was an essential help to this work. I would like to extend my sincere thanks to Gauthier Rotsart de Hertaing for making himself available during this period and for is helpful assistance. I also wish to thank my family and my friends for the happiness and support they bring me. Special thanks to my brother's wife for the spelling and grammar check.

Edwin Desjardin, 06/01/2024

Abstract

Road safety development is still an open field of research. There is a necessity for data acquisition. This work proposes a multimodal sensor for traffic monitoring. The sensor is embedded with a camera, a radar and a lidar. The thesis starts by introducing the roles of the different modalities and continues with the characteristics of all the devices of the system. The whole synchronization principle is explained and optimized to run on a micro-computer. The synchronization uses USB video class protocol, TCP over IP and gptp protocol. After, an example of data processing and combination scheme is proposed. Camera data is processed using YOLOv8 algorithm while radar data is first transformed in Doppler graphs, then filtered and finally processed to extract targets. The Lidar data is processed following a custom scheme. Finally the sensor is used for a dataset creation.

Keywords: Camera, Radar, Lidar, Sensor, YOLOv8, DBSCAN, CFAR, Dataset.

Contents

1 Introduction

1.1 Motivation and proposed contribution

Since the 20^{th} century, the automotive industry has been steadily expanding. Manufacturers began to focus on passenger safety in the 1950s with the development of airbags. Since then, new safety systems have been created, like ABS (Anti-lock Braking System), which helps drivers keep control of the vehicle when breaking. With the massive apparition of electronic devices, came the ESC (Electronic Stability Control). The system prevents the car from skidding by compensating for the lack of grip with specific braking.

Research has continued to be conducted on ADAS (Advanced Driver Assistance Systems). These systems include adaptive cruise control, lane-keeping assistance, blind-spot monitoring, and automatic emergency braking. This development in the automotive industry is linked to emerging technologies in other domains as well. For instance, machine learning has become a major field of research in many industries. The automotive sector is no exception. AI-powered sensors, cameras, and radar systems continuously monitor the vehicle's surroundings, enabling real-time analysis of potential hazards. The goal of AI algorithms is to identify objects, pedestrians, and other vehicles, allowing the vehicle to take preventive actions or provide warnings to the driver. Thus, there is a need for research on sensor fusion schemes and algorithms.

Other crucial points for drivers' safety are the road improvements. On the Walloon infrastructure website [41], one can read (translated): "The studies carried out cover public lighting, traffic lights, radar, tunnel equipment, road signage, safety barriers and all other intelligent transport systems (carpool detection/control, devices for interaction with connected vehicles, etc.). Road equipment also includes road markings, vertical signs, restraint systems such as guardrails, noise barriers and weighting stations."

This work aims to help in this road security research by providing a synchronized multimodal sensor for traffic monitoring. The proposed sensor modalities are camera, radar and lidar. We will also provide a dataset of traffic scenes recorded with the proposed sensor as a second contribution. With these contributions, the goal is to allow further research on applications such as intelligent traffic lights, accident detection systems and traffic control.

Each sensor type (camera, radar, and lidar) provides unique and complementary information about the environment. Cameras capture color images, enabling object recognition and scene understanding. Radar measures velocity and distance, making it ideal for detecting moving objects. Lidar, on the other hand, delivers precise 3D point cloud data, allowing for accurate mapping of the surroundings and the identification of objects based on their shape and size. By combining these modalities, safety systems should achieve a more comprehensive and reliable perception of the environment.

Moreover, multimodal datasets offer redundancy, which is crucial for safety applications. If one sensor encounters issues, such as occlusion for camera due to adverse weather, the other modalities can compensate and provide a more accurate view of the surroundings. This redundancy enhances the robustness and reliability of systems, reducing the risk of accidents caused by sensor failures.

1.2 Roles of different modalities

Our sensor comprises three modalities, a camera, a lidar, and a radar. Each one has a different role in the system. Thus, it is interesting to compare the purposes of each modality, in order to justify their necessity.

- Camera: The camera is used to capture scene surroundings. The data can be processed to detect and recognize objects such as vehicles, pedestrians, cyclists, traffic signs, and lane markings.
- Lidar: The lidar provides a three-dimensional, high-resolution map of the environment using laser pulses. This provides precise distance and depth information, enabling accurate 3D mapping of the surroundings. The lidar also add information about the object's size and shape.
- Radar: The radar gives information about an object's distance and radial speed. It can operate effectively in various weather conditions, including rain, fog, and snow, since radio waves are less affected by environmental factors compared to cameras and lidars.

These sensors serve complementary roles and are all three necessary to provide a comprehensive and robust perception of the environment. The sensor's applications can combine the strengths of these modalities to improve object classification and tracking. For example, the radar can detect the presence of an object, while the lidar can provide precise shape and position information for that object. As explained, this traffic monitoring sensor aims to help safety research. Redundancy is a key aspect of ensuring safety. Having multiple sensor technologies operate on different principles can help cross-verify the data and mitigate the impact of sensor failures or limitations. If one sensor encounters difficulties, the others can still provide essential information. As mentioned previously, a radar can work effectively in heavy rain, while a lidar and camera may struggle due to water droplet interference. On the other hand, a lidar can provide higher-resolution data and is better suited for clear weather conditions.

Some of the following figures are made using Open3D[1] python library and by using graphical content from Flaticon[2].

1.3 Starting point

This work is the follow-up of previous research at Uclouvain. One can cite; Duflot [11], Rotsart de Hertaing [17], Ledent [18], Bolteau [25]. Thus this work does not start from scratch. The starting point is a bimodal sensor combining a synchronized camera and radar. It implies that the choice of the camera and radar models doesn't depend on this work and has already been argumented in the previously cited thesis.

The principal bottleneck of this system from previous work was the inaccuracies in distance estimations. The lidar will provides more accurate distance measurements.

[1] https://www.open3d.org/
[2] https://www.flaticon.com/

However, the choice of the lidar device was preceded by investigations about the lidar's market. There are two types of lidars, time of flight (TOF) lidars, and Frequency Modulated Continuous Wave (FMCW) lidars. The main difference is that FMCW lidars directly provide speed information while tof lidars only provide distance information. Even though some companies announce manufacturing of FMCW lidars on their website, none were found in open sale, or at least none for distance range beyond a few meters. For TOF lidars, there are only few references in open sales for the distance range needed. Livox proposed a few interesting models and the selected one is the Livox Hap TX. The complete sensor is managed by a micro-computer. Here, we use a *Jetson Nano*® micro-computer. The characteristics of the devices will be detailed in the following section.

2 Devices characteristics

The main devices used in the system are:

- Camera: ELP-USBFHD04H (AR0330)
- Radar: RFbeam K-MD2
- Lidar: Livox Hap Tx
- Micro-computer: *Jetson Nano*®

2.1 Micro-computer characteristics

The micro-computer used in this thesis is the *Jetson Nano*®. It is developed and manufactured by *Nvidia*® and is sold as a powerful compact computer designed for neural network computation. The *Jetson Nano*® has a very low power consumption, ranging between 5 to 30 watts. The CPU is a quad-core ARM A57 cadenced at 1.43GHz. It is boosted by an integrated 128-core Maxwell GPU. The OS is a dedicated version of linux installed on a SD-card.

Figure 1: *Jetson Nano*®

2.2 Camera characteristics

The camera reference is ELP-USBFHD04H. It is available at [23]. The camera is composed of a 2 megapixel CMOS sensor named "AR0330". The maximum resolution is 1920x1080 pixels with a maximum frame rate of 30fps. The connectivity is compatible with USB2.0 protocol. The camera is directly supplied in power via USB. The camera needs at most 190mA at 5V, thus, the maximum power consumption is $0.19 * 5 = 0.8W$. There are multiple output formats available; YUV, MJPEG or MPEG-H264. The advantage of this model is the automatic exposure management. It allows the camera to be easily used in different luminosity conditions. There are different parameters that can be adjusted, including, brightness, contrast, saturation,

hue, sharpness, gamma, white balance or back-light contrast. Everything can be controlled via UVC (USB Viedo Class). Camera specifications are summarized in table 1.

Figure 2: ELP Camera

2.3 Radar characteristics

The radar reference is the K-MD2 device, sold by RFbeam at [37]. This is a FMCW (Frequency Modulated Continuous Wave) radar operating at 24GHz. The radar measures distance and velocity of objects in its' field of view. The working principle of a FMCW radar has already been developed by François Ledent. The curious reader is thus invited to consult [18] for further explanation. The maximum detection ranges is between 0 and 250 m with a 1m precision. The radar is capable to detect speed of $\pm 130 Km/h$ with $1 Km/h$ precision. The field of view is defined by ± 9.1 (elevation), ± 16.4 (azimuth) with a $0.1°$ precision. These characteristics are the maximum reachable values for this radar. However some parameters, choices, and trade-off have been made by Ledent in [18] that leads to lower performance. The radar specifications are summarized in table 1.

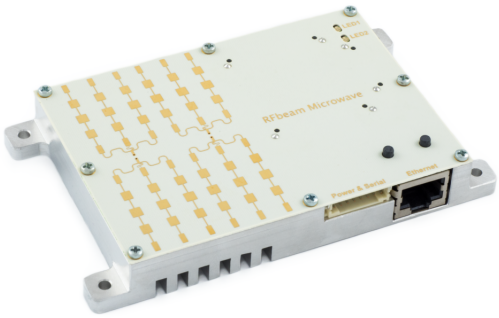

Figure 3: K-MD2

2.4 Lidar characteristics

The lidar used in this work is the *Livox Hap Tx* from Livox. It can be directly purchased from the Livox website [35]. It is visible in figure 4. The device is embedded with an eye-safe laser of safety class 1 according to IEC60825-1:2014 standard. Class 1 represents lasers that are safe for eyes under any normal use [33]. Its' wavelength is 905 nm.

Figure 4: Livox hap Tx [35]

The detection range is up to $150m$ for objects with at least 10% reflectivity. The field of view (FOV) is 120° horizontally and 25° vertically with an angular resolution of 0.18° horizontally and 0.23° vertically. Communication with the external world is achieved with a $100Base-T$ Ethernet connection. It provides $100Mbit/s$ maximum data rate in each direction (full-duplex) using only two pairs of wires, one for each direction. The internal network interface allows for generic Precision Time Protocol (gPTP) which will be described and applied later for synchronization.

The lidar can be supplied with 9 to 18 volts DC. It can thus be directly supplied by the 11V battery used by the initial camera-radar system, without any conversion needed. The typical power consumption is 12W. At startup, it can be up to 26W, but it can be significantly higher at low temperature if the glass heater is needed. In window-heating mode, power consumption may increase by 10W. A complete description of the lidar's power consumption versus temperature is available in [34].

The package size is 105 x 131.6 x 65 mm for a 1120 g weight. It is IP67 rated. It is completely protected against dust and water until 1m immersion for 30 minutes [32]. That is sufficient for our outdoor application.

A summary of the devices characteristics is available in table 1.

	Camera	**Radar**	**Lidar**
Resolution	320x180, 320x240, 640x360, 640x480, 1280x720, 1920x1080	256x256 [FFT_size]	Not applicable
Data format	Mjpeg, YUYV	Raw signals, Doppler graphs	Points
Frame rate	30, 25, 20 ,10, 5 [fps]	12 [fps]	452000 [pointsps]
Field of view Azimuth x Elevation	30x17[°]	33x18[°]	120x25[°]
Interface	USB	Ethernet	Ethernet
Power voltage	5[V]	10-16[V]	9-18[V]
Power consumption	1[W]	9.6[W]	12[W]
Price	∼45€	∼1300€	∼1500€

Table 1: Devices characteristics

3 Lidar's integration

This section will introduce the lidar into the system. As the lidar is a major contribution of this work, more details will be dedicated to it.

3.1 Working principle

The chosen lidar is called a time of flight (tof) sensor by opposition, due to the frequency modulated continuous wave (FMCW) sensors. The working principle of FMCW sensors has been detailed in [18] by François Ledent. Tof lidar means that the data is obtained by measuring the time between a laser pulse emitted and it's reflection returned by the target's surface. The principle's equations are simple:

$$v = \frac{d}{t} \tag{3.1}$$

With the speed of light, d the distance traveled by the light in a time t.

$$d = ct \tag{3.2}$$

$$r = \frac{d}{2} = \frac{ct}{2} \tag{3.3}$$

As c is known, the challenge is to find an accurate estimation of t. In the case of the Livox hap, the estimation scheme isn't publicly provided by the manufacturer. A simple way to achieve that could be to increment a counter at each clock cycle until reflection is detected. There are multiple challenges for this estimation and multiple techniques available in the literature. The noise from sunlight is also a challenging problem. Curious reader can look at [9] for further explanation.

Once r is known, the laser is aimed at a different direction of the field of view. This direction is represented by ϕ and θ, the azimuth and elevation angles. The coordinate points are then stored and sent every 96 points. The Cartesian coordinates can be derived from spherical coordinates using the following equations:

$$x = r \sin\theta \cos\phi \tag{3.4}$$

$$y = r \sin\theta \sin\phi \tag{3.5}$$

$$z = r \cos\theta \tag{3.6}$$

3.2 Communication protocol

The communication between lidar and computer uses User Datagram Protocol (UDP). It is a transport layer protocol with the following characteristics:

- Used over Internet Protocol (IP)
- No guarantee of data delivery
- No automatic re-transmission in case of data loss

- No guarantee of arriving order

Livox Hap uses a communication protocol over UDP. The complete protocol that includes configuration commands is available at [27]. Here, we are going to focus on the point clouds transmission. The packet format for data transmission is described in figure 5.

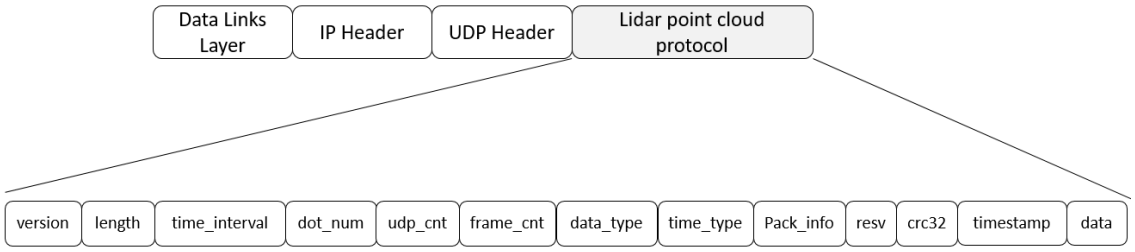

Figure 5: communication protocol packet structure [27]

The Header is made of different fields:

- **version, 1 byte**: Package protocol version.
- **length, 2 bytes**: Length of the complete UDP data field.
- **time_interval, 2 bytes**: Acquisition time of the 96 points in the packet. Difference between the timestamp of the last point and the timestamp of the first point in the packet.
- **dot_num, 2 bytes**: Number of points contained in this packet. Here, it is always 96.
- **udp_cnt, 2 bytes**: Counter of UDP packets. It is incremented by one for each UDP packet. As UDP does not ensure packets delivery order, this is useful to reorder packets at receiving end. With 2 bytes, this counter can go up to 2^{16} which means approximately a reset every 150s. Thus, there is ambiguity if the UDP packet takes more than 2 minutes and 30 seconds to go from the lidar to the computer. In practice, such a delay never appends and moreover, the ambiguity could be removed easily using data timestamps. However it is still useful to determine if there is packet loss.
- **frame_cnt, 1 byte**: Not used for Livox Hap. Set to 0.
- **data_type, 1 byte**: Type of the transmitted data.
  - 0: IMU data. Inertial Measurement Unit represents information returned by the lidar's gyroscope. This is useful when installed on moving objects, such as vehicles.
  - 1: Point cloud data. Point cloud with each coordinate coded on 32bits. This is what we use here.
  - 2: Point cloud data. Point cloud with each coordinate coded on 16bits.
- **time_type, 1 byte**: Type of timestamp used.

  - 0: Timestamps based on lidar's internal clock.
  - 1: Timestamps based on master clock from gPTP synchronization. This is what will be used here. The detail of the synchronization is explained in subsection 4.3.

- **pack_info, 1 byte**: Not used here.
- **reserved, 11 bytes**: Reserved.
- **crc32, 4 bytes**: Timestamp and data segment check code created using crc-32 algorithm. This can be use to check integrity of the data.
- **timestamp, 8 bytes**: Timestamp of the first point of the data. Here, we will include the unix time, based on the synchronization described in 4.3.
- **data, 1344 bytes** Point cloud data formatted as concatenation of 96 points containing each, in this order:
  - X coordinate [mm]: 4 bytes.
  - Y coordinate [mm]: 4 bytes.
  - Z coordinate [mm]: 4 bytes.
  - Reflectivity: 1 byte. Reflectivity of the surface.
  - Tag information: 1 byte. Information about confidence of the point.

3.3 System level modifications

As mentioned in a previous section, the lidar's communication is achieved through Ethernet. The radar also uses Ethernet communication, thus, we need a way to connect multiple Ethernet devices to the same computer for acquisition. The camera isn't concerned because its' communication with the computer is achieved through USB. The chosen solution is the use of an Ethernet switch. It allows us to extend the network to multiple devices. The criteria for the choice of the switch are:

- At least 3 Ethernet ports (one for each device including the computer).
- Able to handle the data rate of the radar and the lidar.

The chosen model is a TL-SG105 from tp-link visible on figure 6. It is a gigabit switch that can handle data rate until $1Gbit/s$ on each of the 5 Ethernet ports provided. It is also plug and play without any configuration needed. Even though a 3 port switch would be enough, 5 port models are more common to find. It also allows further upgrade of the system. Finally, this model was available without delay at a reasonable cost. At purchase time (october 2023), its price was around 25€.

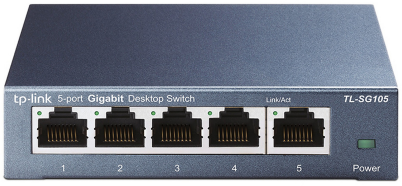

Figure 6: TL-SG105 [38]

The switch needs a 5 Volts DC power supply. Thus, a buck converter is needed if we want to power it with the 11V battery. The system is now represented in figure 7.

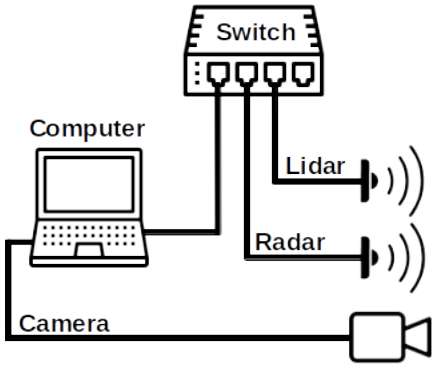

Figure 7: System's routing

4 Synchronization principle

The data generated by a sensor is driven by its internal clock. Moreover, data from different sources may not be inherently synchronized due to variations in their sampling rates, latency, or clock drift. In our object classification scenario, the camera, the radar, and the lidar are independent devices, each one with its own clock. Thus, data needs to be synchronized in a way that it shares the same temporal basis. The goal is to know, the exact acquisition time, for every picture taken by the camera, for every radar frame, and for every lidar frame.

In this section, we will introduce the synchronization principles of the system. The camera and the radar synchronization schemes are the result of François Ledent's work. Thus, only the concepts necessary for understanding will be introduced here. The interested reader is invited to read [18] for more detailed explanations. Lidar data synchronization will be explained in full detail, as it makes a major contribution to this work.

The choice has been made to base the synchronization on linux system's "CLOCK_REALTIME". It is based on POSIX epoch, 00:00:00 on january 1, 1970 [39]. The goal is to find at which value of this clock was taken the picture of the camera, the frame of the radar and the frame of the lidar.

4.1 Camera synchronization

In this section the synchronization of the camera's data will be introduced. For the camera, it is pretty simple. The USB video class protocol used to exchange data with the camera provides a timestamp with each data buffer. This timestamp correspond to the time at which the first byte of the buffer has been written. It is based on the computer's "CLOCK_MONOTONIC". This is a clock from the system that continuously measures the time from an unspecified point in the past [39]. Therefore, the absolute time of the buffer can be obtained with equation 4.1, where t_{stamp} is the absolute time we are searching for and $buffer.timestamp$ the time provided by USB video class.

$$t_{stamp} = CLOCK\_REALTIME - CLOCK\_MONOTONIC + buffer.timestamp \quad (4.1)$$

4.2 Radar synchronization

For the radar, it is much more complicated as communication is achieved through Ethernet via TCP (Transmission Control Protocol) over IP (Internet Protocol). IP is responsible for the internet layer while TCP is responsible for the transport layer. Through the use of specific headers, TCP ensures the integrity and order of segments. At receiver end, the protocol reorders segments such that it corresponds to the order they where sent by. The drawback is that integrity means that in case of loss, the protocol asks the transmitter for a new copy of the missing segment and thus introduces a variable delay.

In order to understand how the timestamps are inserted in the data-frames, we first need to introduce how the connection between the radar and the *Jetson Nano*® is established. An adapted scheme from Ledent [18] is available in figure 8.

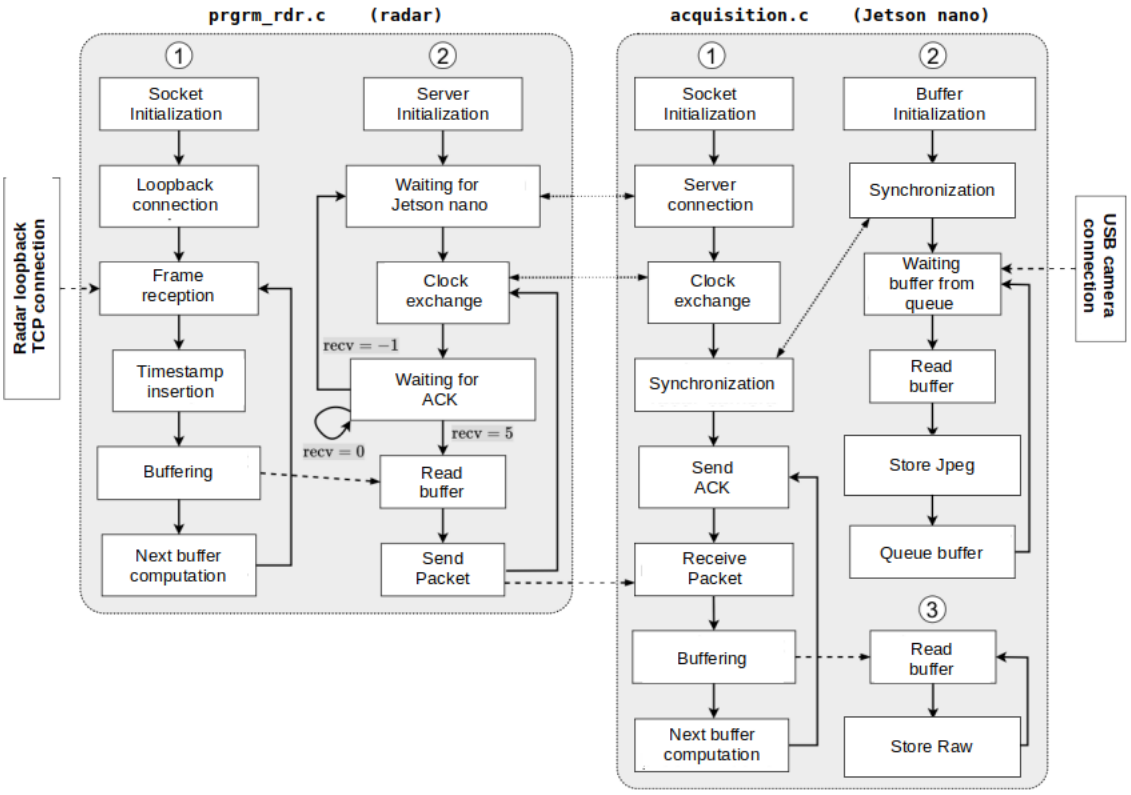

Figure 8: Ledent's implementation of devices communication

The whole interaction is managed by two scripts, *prgrm_ rdc.c* in the radar and *acquisition.c* in the*Jetson Nano*®. These scripts are divided into a total of five threads. The first one, TR1 (thread radar 1), inside the radar, creates a connection with the server called *loopback*. This server is a TCP connection between the internal components of the radar. The second one, TR2 (thread radar 2), and the third one, TJN1 (thread *Jetson Nano*® 1) are responsible for the connection between the radar and the *Jetson Nano*®. The fourth one, TJN2 (thread *Jetson Nano*® 2) manages the connection with the camera and the storage of the corresponding data. Finaly, TJN3 (thread *Jetson Nano*® 3) stores the data from the radar into the SD-card.

Now that the radar and the *Jetson Nano*® are able to communicate, let's talk about how the synchronization works. In order to be as accurate as possible, timestamps must be inserted as close as possible to the hardware sensor that acquires the data. The problem is, the radar only knows its' own internal clock and not the reference one, "CLOCK_REALTIME". Therefore, we need a reference time t0 where we know both the value of the radar's clock and "CLOCK_REALTIME". Once this reference is known the timestamp can simply be find following equation 4.2.

$$t_{stamp} = CLOCK\_REALTIME_{t0} + CLOCK\_RADAR_{frame} - CLOCK\_RADAR_{t0} \quad (4.2)$$

To obtain this reference t0, a clock exchange scheme has been created by François Ledent [18] and is shown in figure 9.

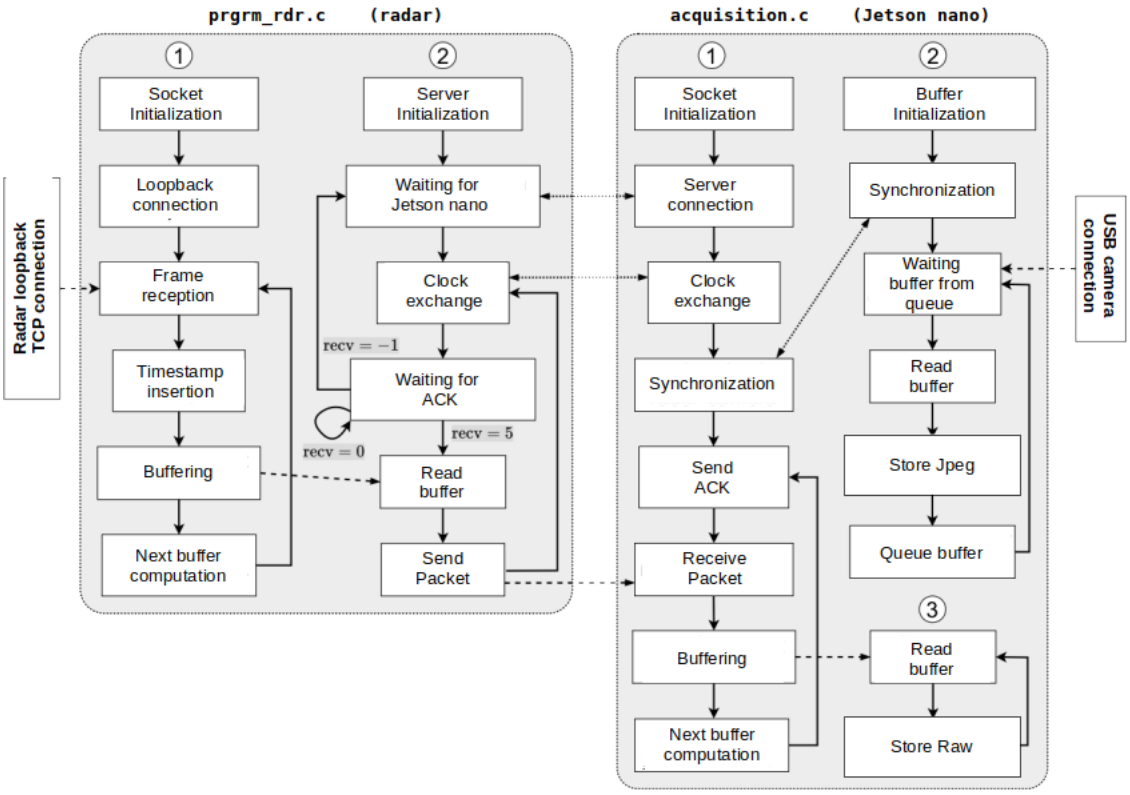

Figure 9: Ledent's implementation of clock exchange translated from [18]

The method is the following. Once the connection between the radar and the *Jetson Nano*® is established, the micro-computer sends what we call a "clock request". The radar responds by sending back its internal clock. Thus, the *Jetson Nano*® knows the value of $CLOCK\_RADAR$ at a certain time t0 located between the time when the "clock request" was sent and the response was received. A simple algorithm repeats this process of "clock request"/response until the complete exchange time is below a certain threshold. Here $200\mu s$ was arbitrary chosen as acceptable. This threshold can not be arbitrary low because of the physical path delay in the cable. If we suppose that the TCP exchange takes the same time from the *Jetson Nano*® to the radar than the opposite, then, the error on $CLOCK\_RADAR_{t0}$ is smaller $\frac{200}{2} = 100\mu s$.

4.3 Lidar synchronization

As explained in the previous section, the camera and radar are synchronized based on the unix clock. This clock measures the time elapsed since 00:00:00 UTC on 1 January 1970 [40]. We thus need to synchronize the lidar's data with this clock. As mentioned in the communication protocol, the lidar point cloud protocol already includes 8 bytes for the timestamps. The synchronization will be achieved using gPTP (generic Precision Time Protocol). If this synchronization was not implemented the lidar would have filled this timestamps by its internal clock based on the time it is powered on.

4.3.1 generic Precision Time Protocol(gPTP)

The generic Precision Time Protocol, is a network protocol used to precisely and accurately synchronize clocks in a network. All the characteristics are specified in the IEEE 802.1AS standard [29] [30]. The working principle is made of multiple steps. The first one is the estimation of the link delay between all the pairs of neighboring nodes in the network. The link delay represents the physical delay in the link, here the cable, between two devices. For a pair of nodes the estimation of the link delay is represented in figure 10

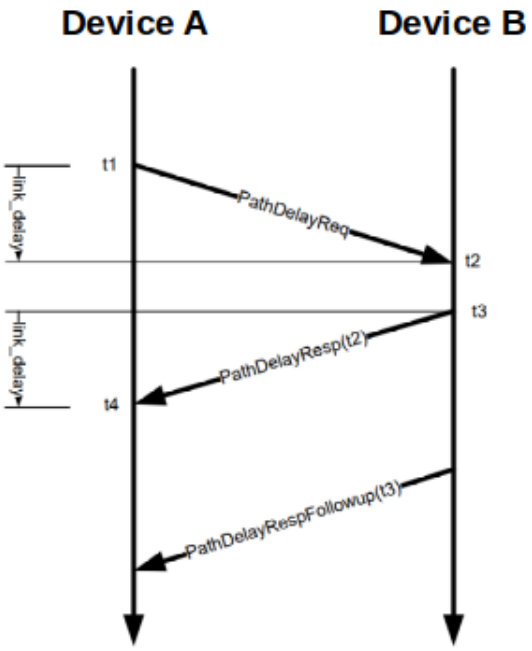

Figure 10: gPTP link delay computation [30]

Device A sends a path delay request at time t1. Device B receives this request at time t2 and responds with the value of t2. This response is sent at time t3 and received at time t4 by device A. Then, Device B sends another message, called followup, to device A specifying T3. Here, it is important to understand that t1 and t4 are time based on the clock from device A while t2 and t3 are time based on clock from device B. Here we made the assumption that the link delay is the same from A to B than from B to A.

Thus, the link delay can be found using the following:

$$link\_delay = \frac{(t4 - t1) - (t3 - t2)}{2} \tag{4.3}$$

Operation on clock times have to be done carefully not to mix times from different references. Here we first subtract times from the same reference clock and then subtract the resulting elapsed times.

This operation needs to be repeated for all the pairs of neighboring devices in the network that uses the gPTP synchronization. In our case, there are only three nodes in the gPTP domain.

They are the computer, the network switch and the lidar. The situation is represented in figure 11.

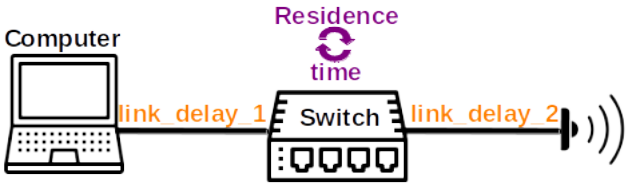

Figure 11: gPTP domain

In this figure, we see our gPTP domain. The link delays and the residence time are represented. The residence time represents, for a message, the delay inside the switch between the time the message arrives in the switch and the time the message leaves the switch. It is important to mention that the switch needs to respond to its' neighboring nodes to compute the link delays. Thus, we need a switch that is compatible with gPTP, which is the case for the one we chose.

Now that all the link delays are known, let us see how to use those delays to synchronize all the ending-nodes of the network. First, we need to define a master node from which the clock will be the reference in the network. As we want to synchronize our system on the previously cited unix clock, the master node will be the computer. Note that the computer needs an Ethernet interface which is compatible with gPTP. As the computer is the master node, all the other ending nodes are the slaves (i.e., the lidar).

The final working principle of the synchronization is represented in figure 12. For simplification purpose, only one switch has been represented, as inour configuration. The general principle for a network with multiple switches can be easily derived by cascading the same principle.

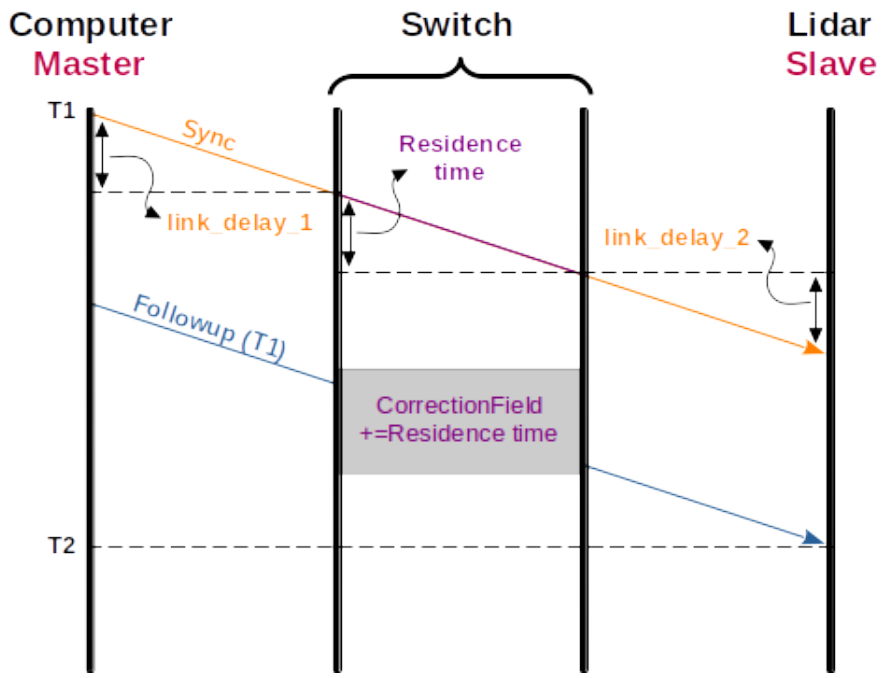

Figure 12: gPTP

The master node sends a sync message to every slave node. This message is affected by all the link delays from the master to the particular slave plus all the residence times of the switches that the message needs to cross. Link delays are known from previous steps but residence times are not. That's why we need a second message called followup that includes T1, the time at which the sync message was sent, as well as a correction field. The role of this correction field is to propagate the sum of the residence time experienced by the sync message. Each switch crossed by the followup adds the residence time of the sync message to the correction field.

At the end, when the slave receives the followup, it can compute T2, the actual clock time of the master and synchronize its' own actions on it. In our case the lidar uses the clock of the computer to add timestamps to the point clouds. These gPTP operations are continuously repeated to ensure the most accurate synchronization and particularly to avoid clock drifts.

5 Jetson nano limitations

After tests, it appears that drops occur in acquisition frame rate when the *Jetson Nano*® is used as a micro-computer. From Ledent [18], we know that the same problem occurs while using a raspberry pi as a micro-computer. It implies that the problem already occurred before the integration of the lidar. This section is dedicated to the analysis, understanding and solving of this problem. For simplifications of the explanation, this section will be limited to camera and radar data only.

5.1 Analysis

First of all, it's important to mention that there is no problem when replacing the *Jetson Nano*® by an Asus laptop with higher performances. The main differences between the *Jetson Nano*® and the laptop are summarized in table 2.

	Jetson Nano®	Asus-G551JK
CPU	Quad-core ARM Cortex-A57 MPCore	Intel core I5
GPU	Arch. Maxwell 128 core CUDA	Nvidia geforce gtx850m
RAM memory	4Go LPDDR4	10Go DDR4
Storage	128Go SD card	256Go SSD

Table 2: Comparison between laptop and *Jetson Nano*®

We need to find which *Jetson Nano*®'s characteristic is the limiting one. After some monitoring of the *Jetson Nano*® while the acquisition is running, different things have been observed. The reader is invited to pay attention to figure 13.

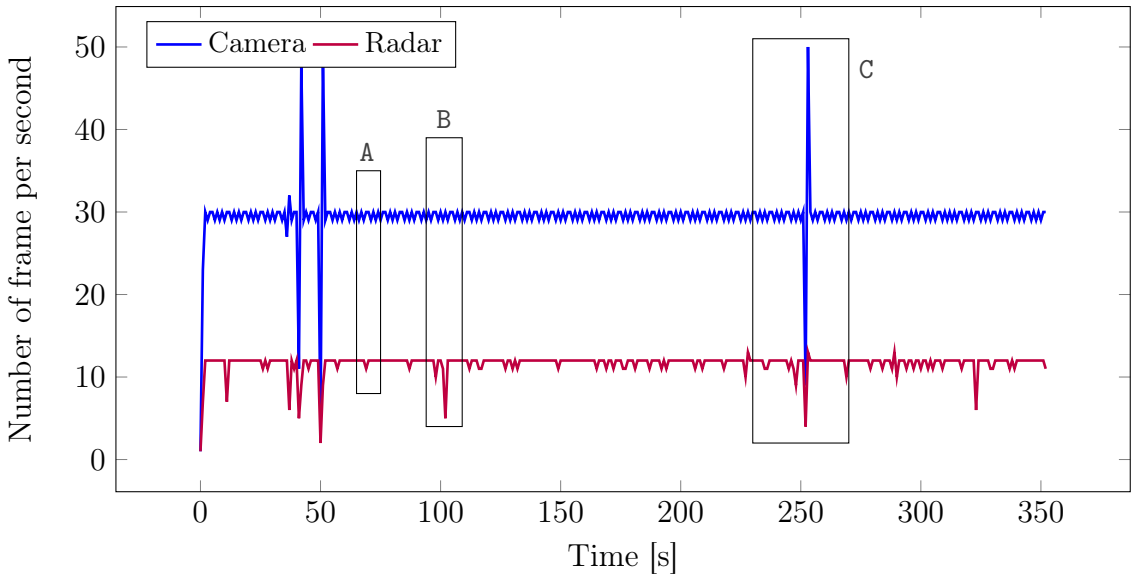

Figure 13: Camera images and radar frame rate versus time while running acquisition on *Jetson Nano*®

In this figure, multiple things are observed. First, we clearly see the drops we mentioned previously as the number of frames per second isn't constant for the radar nor the camera. Also, we can identify recursive patterns. They are represented by the symbols A, B and C. They will be detailed just after. Now let us observe in figure 14 some internal metrics of the *Jetson Nano*® while the acquisition from figure 13 was running.

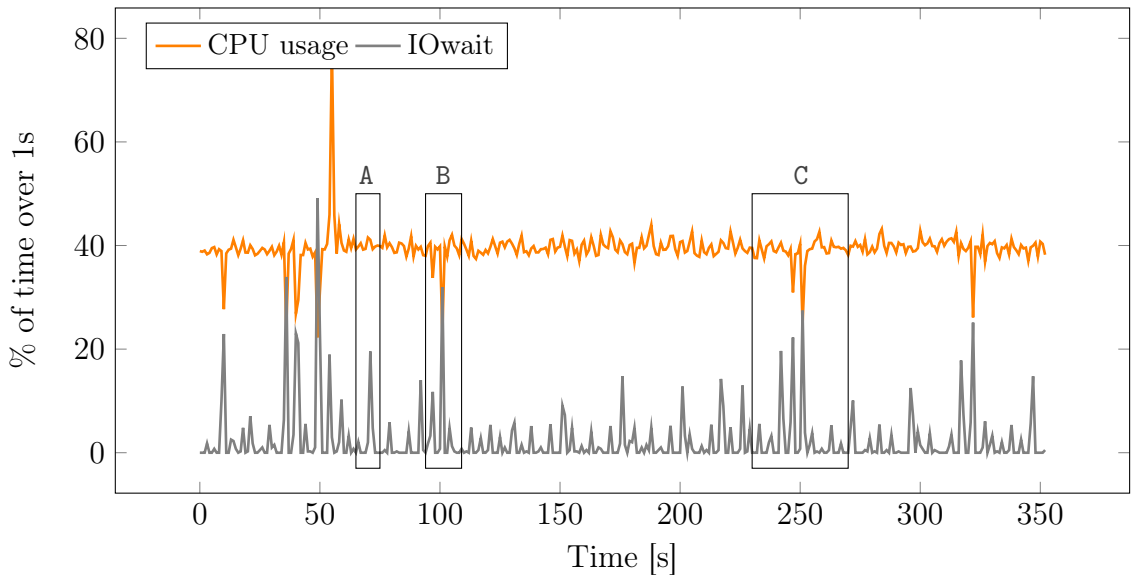

Figure 14: CPU usage and Iowait versus time while running acquisition on *Jetson Nano*®

First, the CPU activity is most of the time below 50%. Moreover, the used RAM memory

(which is not represented here for readability purpose) is approximately constant and equal to 1.75Go which is $\approx 44\%$ of the available RAM memory. We also observe that when acquisition drops occur, CPU activity also drops to sometimes less than 10%. In fact the CPU is waiting for something. Another metric that can be monitored is the %IOWait. It represents the percentage of the time that the CPU is waiting for the disk to handle I/O request [31]. If we monitor this metric while running the acquisition, we observe a huge increase in %IOWait when drops occur. Our attention is therefore focused on the storage. This hypothesis is checked by running the acquisition without saving the data. In this situation, no drops occur. We can conclude that the limiting components is the storage which is here a 128Go SD card.

Let's come back to figure 13 and the different visible patterns.

- **A** represents acquisition when no drops occur. The little fluctuations are due to the way the frames are counted. Thus the number of fps is fluctuating from 29 to 30 for the camera and 11 to 12 for the radar.
- **B** represents acquisition when a drop occurs. This drop is not that big such that only radar frames are affected. The size of a radar frame is bigger than the size of a camera image. That's why radar frames are more affected by those storage drops.
- **C** represents acquisition when a drop occurs, but this time the camera frame rate is also affected. We can observe that just after a drop the number of camera frames increases immediately. In fact, for a mean over a few seconds, there is no loss for the camera. This is due to the number of buffers that are available for the camera in the RAM memory. Even if the SD-card is slow during a few hundreds of milliseconds the camera still saves its data in the RAM memory. Later, when the SD-card is faster than before, the delay is resorbed. This is not the case of the radar because the implementation is made out of three buffers only (this will be detailed later). Thus the radar could only recover from at most one late frame, which is sometimes visible in figure 13 around C.

When Ledent implemented the acquisition scheme between the radar and the micro-computer, he stated that the bottleneck is the communication. Thus, as explained in [18], he implemented this communication with only three buffers. As the goal was real-time processing, the saved buffer is always the last one received. Thus if congestion occurs in the micro-computer, data from previous buffer is discarded.

5.2 Image size parameter

A way to solve or at least reduce this problem is to reduce the amount of data that needs to be stored on the SD-card. With actual parameters, a radar frame size is approximately 786Ko while an image size is 320Ko. With 12 fps for the radar and 30 fps for the camera, the data rates are respectively $12 * 0.786 = 9.432[Mo/s]$ and $30 * 0.320 = 9.6[Mo/s]$. The *Jetson Nano*® needs to handle $9.432 + 9.6 = 19.032[Mo/s]$.

Unfortunately, decreasing the radar datarate by reducing radar resolution would degrade too much the performance. However the resolution of camera can be adjusted. For now, the resolution was set to full HD (1920x1080 pixels). Let's see how reducing this resolution impacts

our storage drops. Table 3 summarize the tested resolutions chosen among the available resolutions from the camera.

Resolution	# pixels	Image size [Ko]	Needed writing speed [Mo/s]
1920x1080	2 073 600	320Ko	19.032
1280x720	921 600	142Ko	13.698
640x480	307 200	47Ko	10.851

Table 3: Tested camera resolution

The chosen metric for the results is the average radar frame rate. They are available in table 4. As the camera frame rate is almost constant, it is not included here.

Resolution	Radar frame rate [fps]
1920x1080	11.646
1280x720	11.669
640x480	11.768

Table 4: Achieved radar frame rate versus the camera resolution

We see that, as expected, the less data the *Jetson Nano*® has to store on the SD-card, the better is the radar frame rate. However, if we look at figure 15, we see that even though there are less drops, still some data is lost for the radar frames.

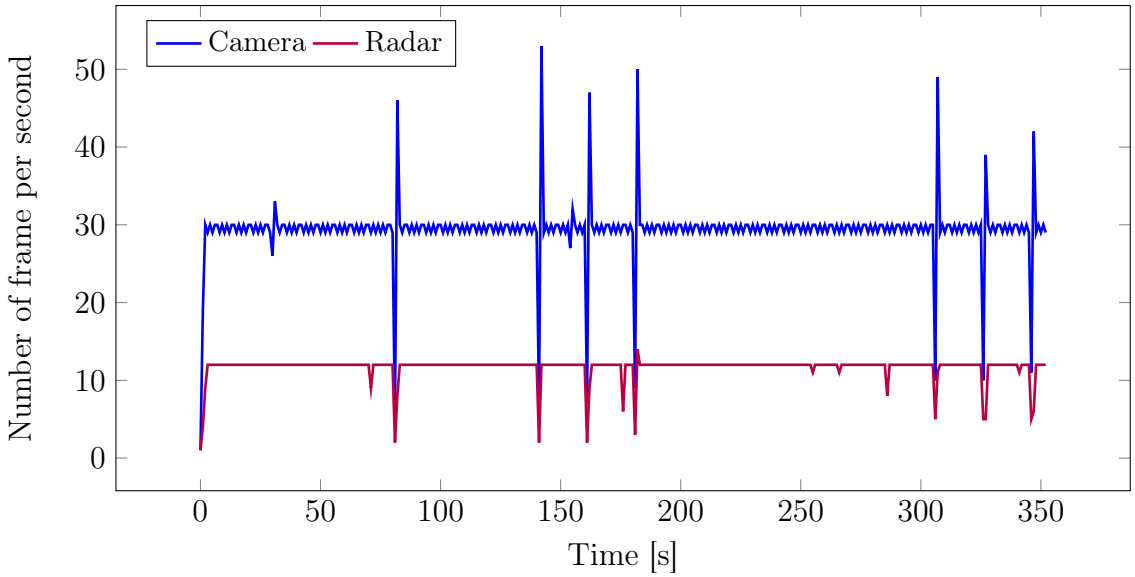

Figure 15: Camera images and radar frame rate in function of the time while running acquisition on *Jetson Nano*® with a camera resolution of 640x480 pixels

5.3 Queue size parameter

The purpose of this work is to provide a dataset for further algorithm development. It is not to provide a system capable of real-time analysis. As explained before, while Ledent developed the synchronisation scheme between the radar and micro-computer, he stated that the bottleneck was the communication. This implied that no late could be resorbed and thus, buffering was not an option for real-time analysis purpose. Here, we can relax this condition as we don't need real-time processing.

This section suggests analyzing a queue extension for radar frames buffers. The SD-card produces writing drops but we can try to take the maximum of the mean writing rate. For that purpose, we are going to analyze the impact of the buffer's queue size on the average saved radar frame rate. The implementation has now been modified to use an easily editable number of buffers. This modification is at the cost of a higher ram memory usage which stores the data while waiting for it to be written on the SD-card. Results are visible in figure 16.

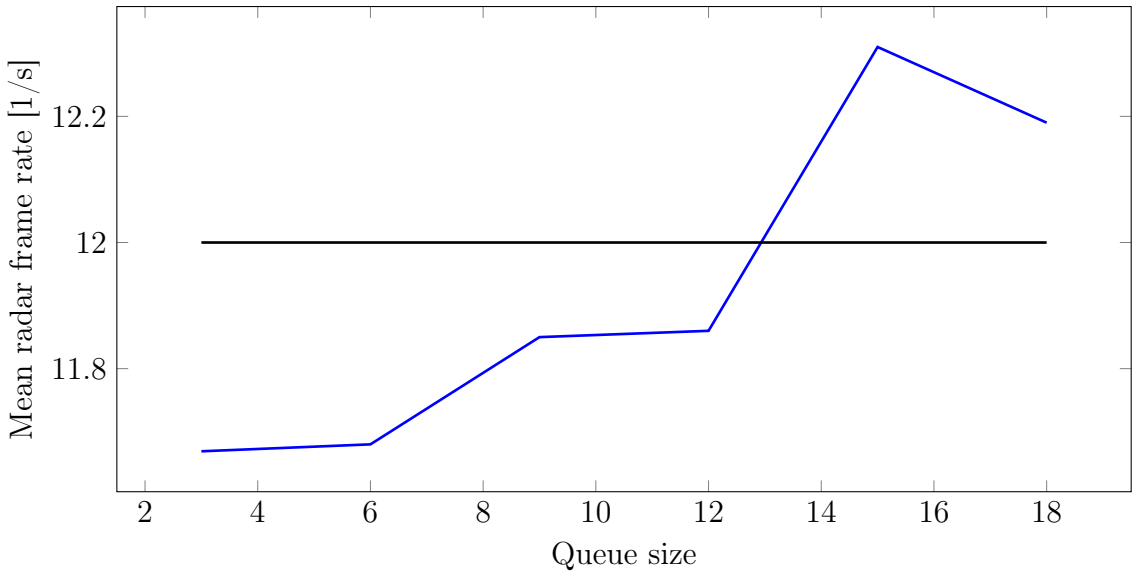

Figure 16: Mean radar frame rate in function of the queue size with a camera resolution of 1280x720 pixels

The tests for the queue size were made from a mean value over 360 seconds. Longer tests could be made to obtain more precise values but as a queue size of 15 seems to be enough, let's keep a margin and choose a queue size of 18. Note that the blue curve should align to 12 fps without exceeding it. This is due to a non-ideality in the way fps are monitored.

5.4 Real-time analysis

As mentioned in the previous section, this work doesn't aim to provide a real-time computation system. Nevertheless, it is interesting to mention the improvements needed to the system to satisfy this constraint.

In fact, the issue here is to store the sensor's data. If we think about the previous section, we just concluded that the RAM memory can be used as a buffer to compensate for storage lacks. Thus, we indirectly proved that the RAM memory can handle this amount of data. The use of this system as real-time analysis system is thus not compromised as far as we do not need to store all the data in the main storage. All the sub-processes, for example, object detection, could directly access data through pointers in the RAM memory. It is still possible to use it as a real-time input for a more complex system.

6 Data processing and results

In this section we are going to introduce the processing of the sensor's data. It aims to show the possibilities of applications and to demonstrate the utility of the lidar in the whole multi-modal system. The data combination scheme between the different modalities will also be proposed.

In this section, the processed data has been recorded with the described system. The acquisition's protocol will be detailed later in the dataset creation section 7.2.

First, in order to better understand the following representations, let's describe the scene that we will observe. The situation is shown in figure 17

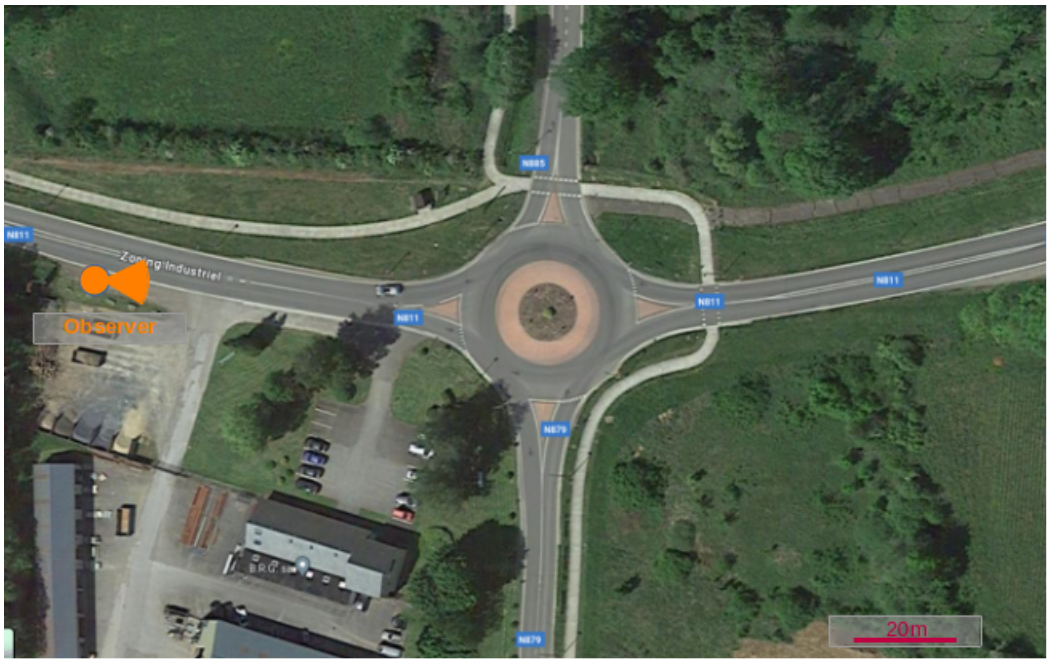

Figure 17: Top view of the scene from google maps

The scene is a two-way road with a roundabout located at approximately 80 meters of the observer. The location is at junction between N811, N879 and N885 in Belgium. Even though the speed limit is 90 km/h, vehicles tend to drive under the limit due to the roundabout. We observe right-hand traffic. Now, let's look at the scene from the observer's point of view. Figure 18 is an image from the system's camera sensor.

Figure 18: observer's POV of the scene

The proposed scheme combines results of independent data processing from the three modalities. Thus the following sections will first describe data processing and then introduce the combination scheme.

6.1 Camera data processing

This section introduces the processing to extract useful information of the data from the camera. A common approach is the use of Convolutional Neural Networks (CNN). There are multiple CNN designs available in the literature. The interested reader can investigate [20] for a complete review. These CNNs are divided in two categories; one stage detectors and multi-stages detectors. For multi-stages schemes, one can cite [4] and [7] that propose Region-based CNNs (RCNN). One-stage schemes are also available, we can cite; You Only Look Once (YOLO)[6] and Single Shot multibox Detector (SSD)[5]. Here we chose the YOLO algorithm in its fifth version, YOLOv8.

6.1.1 YOLOv8

YOLOv8 is provided by ultralytics [3]. This CNN is pre-trained for different classes of object. Here we will focus on classes; bicycle, car, motorcycle, bus, and truck. The model is pre-trained for these classes. Thus, there is no need to train the model and we can directly use it. Moreover, it ensure portability without risk of overfitting.

Here YOLOv8 will be seen as a black box that detects and classifies objects in the images. It provides bounding box, label, and level of confidence for all the detected targets in the image.

[3] https://github.com/ultralytics/ultralytics

The objects with confidence level above a defined threshold are displayed. An example of its output is visible in figure 19.

Figure 19: Example of output of YOLOv8. All the targets are detected. As expected, confidence level of smaller targets are smaller than for bigger targets

6.2 Radar data processing

This section is dedicated to the processing of the radar's data. As well as the explanations of the working principle of the radar, the data processing is detailed in [18]. Thus, this section is limited to an overview of this process. The first step is to obtain the doppler graphs, then filter the result and extract the useful content. The axis of the graphs are limited to the maximum value without ambiguity. These ambiguities appears due to the sampling of the signals and the maximum value is limited by the sampling rate of the radar. A complete explanation is available in [18].

6.2.1 Doppler graphs

Doppler graphs are obtained from the raw data by applying a two-dimension Fourier transform. This can be done using the Fast Fourier Transform (FFT) algorithm. The result is visible in figure 20 adapted from [18].

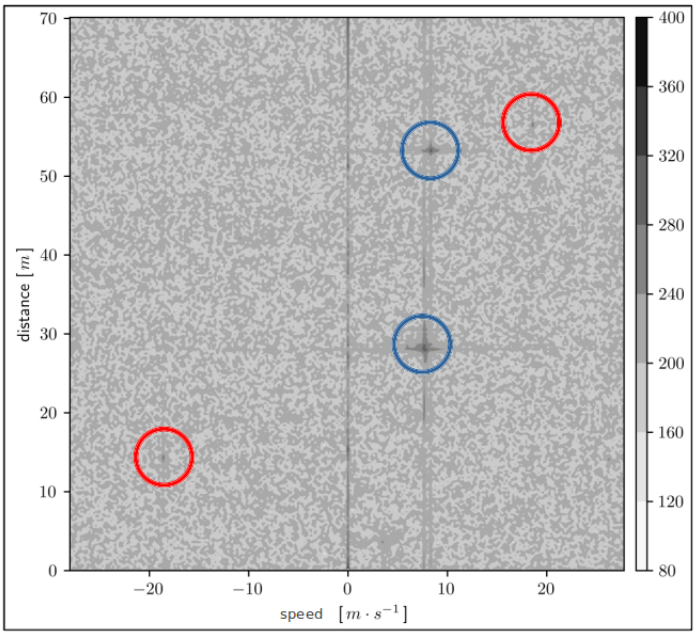

Figure 20: Example of doppler graph in dB

This doppler graph represents a scene with two targets located at 28 and 54 meters. They are circled in blue. It is the norm of the FFT in dB. Distances as the crow flies are represented on the y-axis while the x-axis represents radial speeds.

6.2.2 Filtering

If we look at figure 20, we can see two other less visible local maxima in the FFT. They are located at 15 and 55 meters nd circled in red. These maxima do not belong to targets, they are due to the non-idealities of the radar. Also there is a vertical line of maxima at $0\frac{m}{s}$ which represents the background of the scene. The goal of the filtering is to remove the maxima that do not belong to targets.

The implemented filter is an Exponential Moving Average (EMA) filter. The result of the filtering of figure 20 is visible in figure 21

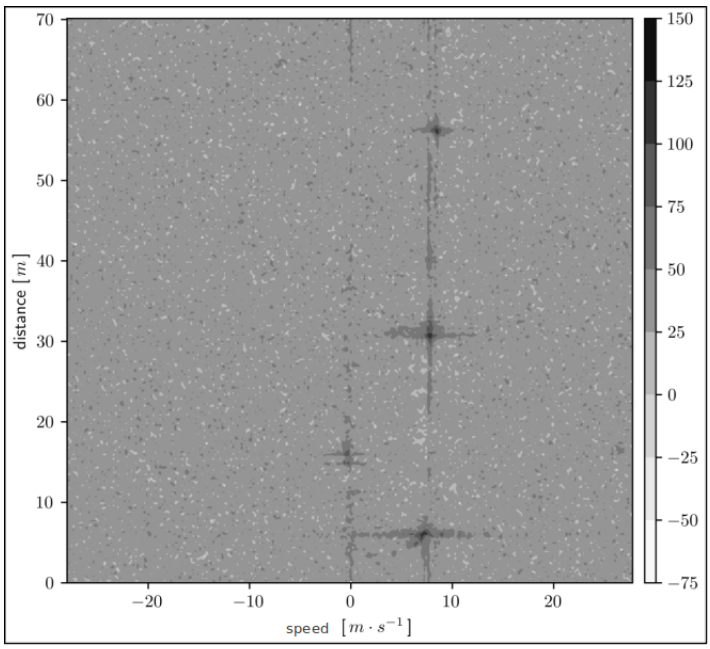

Figure 21: Doppler graph in dB after EMA filtering

As expected, background and points from non-idealities have been removed.

6.2.3 Feature extraction

For the extraction of the useful data, [18] proposes a comparison between the CFAR algorithm and another one based on correlations. CFAR stands for Constant False Alarm Rate. CFAR was chosen over correlations for the purpose of this work as it provides slightly better results. This algorithm is used in the literature, one can cite [1].

CFAR compare the value of a point to the mean of its' neighborhood. A point P is detected as a target if $Doppler(P) > Threshold * Mean(Doppler(Neighbors(P)))$. An example of a result is visible in figure 22.

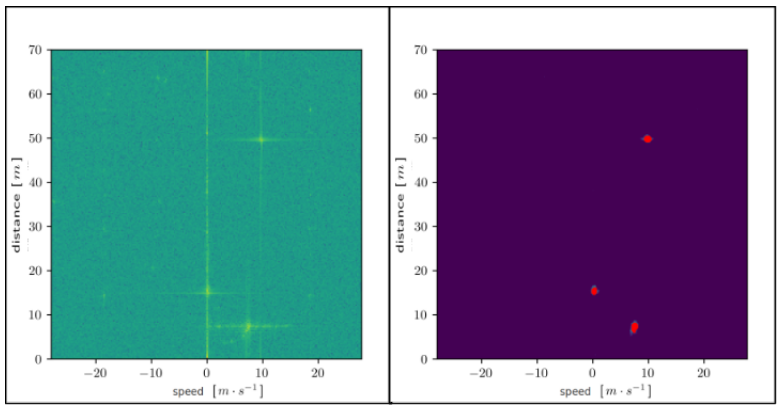

Figure 22: Example of CFAR result

On the left is the original doppler graph. On the right, filtering and CFAR have been applied. Detected targets are represented in red.

6.3 Lidar data processing

In this section, we will define a lidar frame and see the proposed scheme to go from lidar's raw data to target clusters. The presented strategy is the following:

- 1. Remove background to isolate moving objects.
- 2. Filter result to remove noise.
- 3. Cluster remaining points.

6.3.1 Frame definition

Livox Hap uses a non-repetitive scanning pattern. It means that the laser never exactly illuminates the same direction than the one from a previous time. It leads to a better coverage of the field of view but it also has consequences. First, we do not have, for a given direction (θ, ϕ) in spherical coordinates, the update of the distance in this direction every frame. Thus we can not simply obtain moving objects by looking at changing distances in directions, or at least not directly with raw data. Secondly, we need to define a frame. For the camera, a frame is simply an entire image where pixels are aligned as an array. With the lidar, we receive points 96 by 96 continuously.

The choice of the number of point in a lidar frame will influence the resolution but also what we will call motion blur. We could aggregate as many points as we want to increase the resolution of the point cloud. The counterpart is that the elapsed time between the first and the last recorded point also increases. For background, it is not a problem. For moving object in the scene, it creates motion blur.

Let's discuss an example for better comprehension. An observed vehicle is traveling at 90 km/h. We increase the desired number of points to 452000, which is approximately an aggre-

gation of 1s of lidar points. During the acquisition time the vehicle has traveled $\frac{90}{3.6} = 25m$. The vehicle is now visible in the data over 25m more than it's original shape. This phenomenon is represented in figure 23.

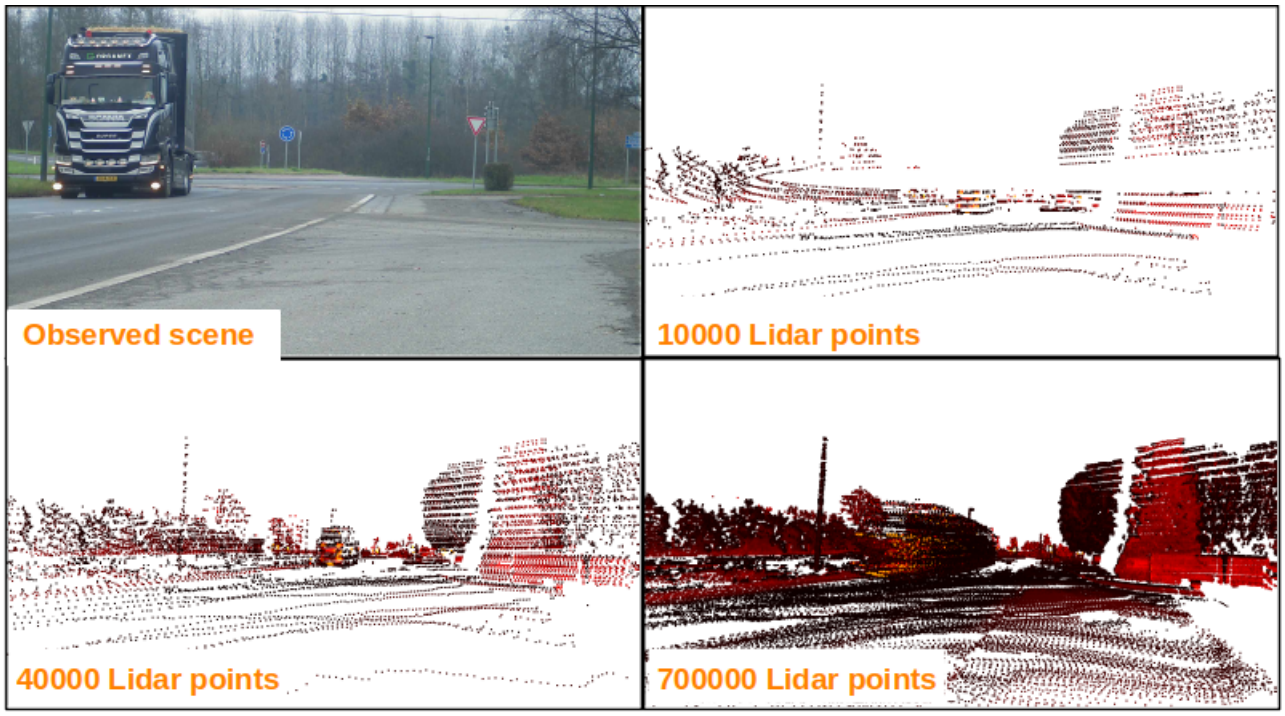

Figure 23: Lidar's data frame representation in function of the number of points

We observe that, as expected, more points implies more detail but also increased motion blur. A trade-off needs to be made. As following post processing may be very time consuming and of high complexity, the arbitrary choice of a frame size matching the radar's 12 fps is made. It leads to $\frac{452000}{12} = 37666$ points per lidar frame.

Note that this represents 12 fps in a discrete point of view where all points are different from the previous frame. Also, as each point has its' own timestamp, we could overlap frame if the application needs more fps. It does not add data, but it makes it easier to represent.

6.3.2 Background

The idea of this step is to suppress all the points of the frame that belong to the background. In order to achieve this, we will first aggregate the frame of the scene without a vehicle to obtain an accurate representation of the background. Then, when processing a frame we will remove all the points that are too close to the background points.

The background of the following picture is the aggregation of 19.25s of the scene without vehicle which represent 231 lidar frames for 8.700.846 points. The result is represented in figure 24.

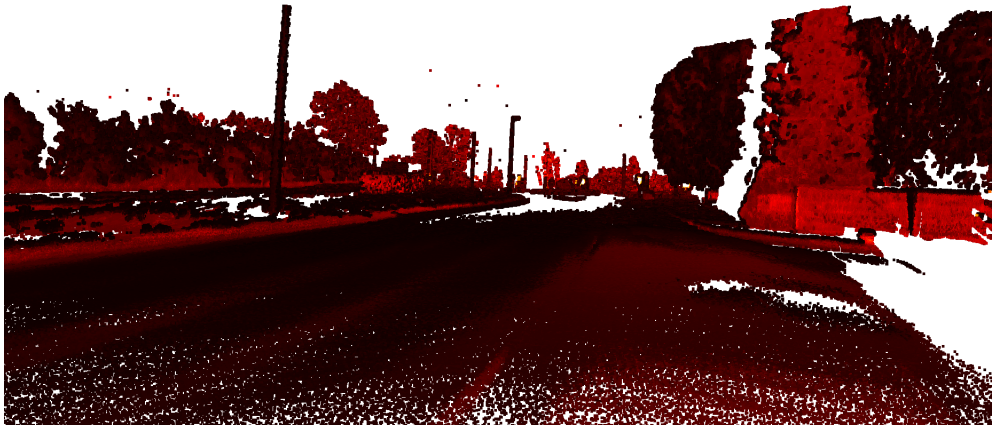

Figure 24: Lidar's representation of the background of the scene

However, it is a rather precise representation of the background, it is pretty big. With this representation, in order to remove the background from a frame, we would need to compute the distance between each of the 37666 points of the frame with each of the 8.7 million points of the background. To avoid this, we will represent the background by voxels. This representation can be seen as a down-sampled version of the background. The 3D scene is divided into small cubes i.e., the voxels. Each voxel represents a part of the 3D space such that all the points of the background are included into a voxel. We can now represent the background by displaying all the non-empty voxels.

Later, points that are to close to those voxels will be deleted. Thus, there is a trade-off in the size of the voxels. Less voxels but bigger ones will need less distance computations to remove background and will be less sensitive to noise. However, if they are too big they may include part of the scene that do not belong to the background and thus later delete points that belong to targets.

Some experiments have been made by trial and error which leads to an arbitrary size of voxels of 300mm x 300mm x 300mm. This size is particular to traffic scenes analysis. The result is represented in figure 25.

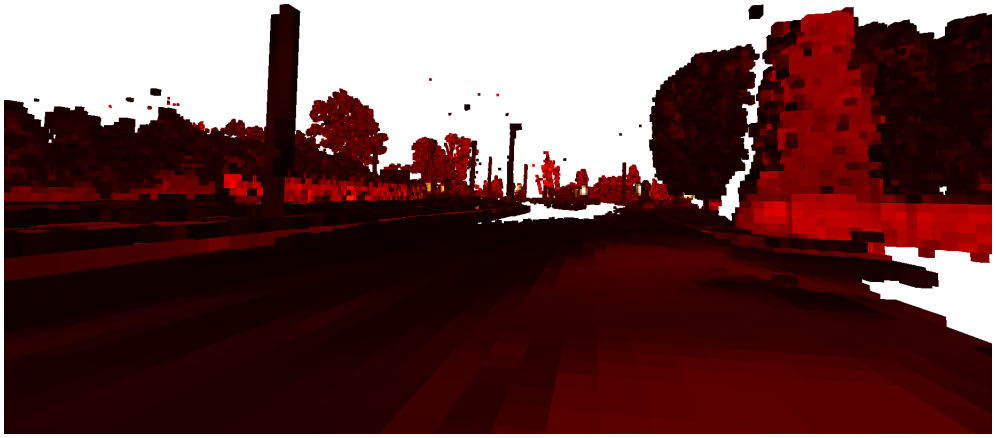

Figure 25: Voxels representation of the background of the scene

The background is then removed from frames. Figure 26 shows the result when removing the background on the frame from figure23. For better visualization, the result has also been converted into voxels.

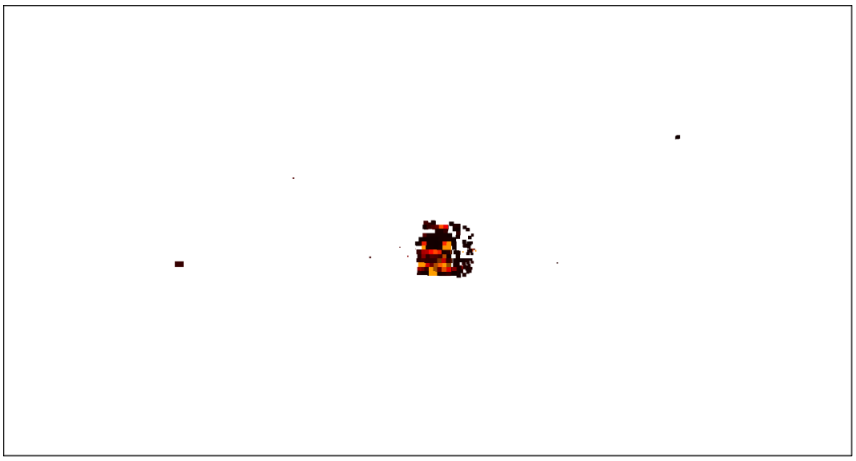

Figure 26: Lidar frame without background

Even though the result seems promising, we can see that some points that do not belong to the target are still visible.

6.3.3 Filtering

In this section, we will remove the noise that remains in the frame after removing the background.

The filtering to remove points that do not belong to a target is done by analyzing the mean distance between the points and their neighbors as a statistical distribution from which we will remove the outliers. This filtering depends on two parameters, the number of neighbor points considered for the mean distances and a second parameter that acts on the filtering threshold. The filtering is done according to the following pseudo-code:

```
average_dist=[]
for each point in the frame:
    distances=Distance from nearest neighbor (point,K)
    mean_dist=mean(distances)
    add mean_dist at the end of average_dist

cloud_mean=mean(average_dist)
std_dev = standard deviation of (average_dist)
treshold=cloud_mean+Param2*std_dev

for each point in the frame:
    if average_dist[point] is lower than the treshold:
        remove the point from the frame
```

First, for each point of the frame, the distances with their k-nearest-neighbors are computed and the mean of those distances are stored in the average_dist list. This list represents the statistical distribution of the mean of the distance between a point and its k nearest neighbors. Then, the mean and standard deviation of this statistical distribution are computed. A threshold is established based on those values and a second parameter. Finally, points that do not satisfy the threshold are discriminated.

The first parameter is K, the number of considered neighbors. The second one is a factor that acts on the threshold which defines the filter's aggressiveness. In order to not suppress a point, K represents the number of close neighbors needed while the other parameter represents how close those neighbors need to be.

The choice of those parameters needs to be discussed. We need to define metrics to optimize by adjusting those parameters. The proposed one is the accuracy. $Accuracy = \frac{(TP+TN)}{(TP+FP+TN+FN)}$ with TP, TN, FP and FN defined as following:

- **True Positive (TP):** Number of points that are deleted that do not belong to targets.
- **True Negative (TN):** Number of points that are not deleted that do belong to targets.
- **False Positive (FP):** Number of points that are deleted that do belong to targets.
- **False Negative (FN):** Number of points that are not deleted that do not belong to targets.

For this parameter optimization, a dataset of points with targets and noise labels are needed to evaluate accuracy of the tested parameters. Manually labeling all the points of thousands of frames would be way too long and tedious. The proposed approach is to define a model of the points that belong to a target and a model of the point that belong to noise. Then, use

those models to artificially recreate a labeled dataset to evaluate the filtering. The following characteristics have been manually observed in lidar's data [4]:

- A scene generally contains from 0 to 6 targets.
- A target generally contains from 5 to 200 points
- A scene generally contains from 0 to 8 noise clusters.
- A noise cluster generally contains from 1 to 3 points.
- Targets are generally more than one meter apart from each other.
- Targets mass centers are generally less than 2m above the ground.

Two models of cluster are derived from those characteristics.

The target's model is defined as a cluster of 5 to 200 points with a centroid z coordinate below 2m. All the points are randomly distributed around the centroid with a distance between points and the centroid affected by the type of the vehicle (randomly chosen).

The noise's model is defined as a cluster of 1 to 3 points with a centroid anywhere in the 3D space.

A complete scene is artificially created by creating 0 to 6 targets and 0 to 8 noise clusters. The difference with acquired data is that every point is labeled with "target" or "noise" to allow evaluation of the filtering. This operation is repeated to create a dataset.

Then, the filtering is applied on this dataset and it's accuracy is computed. This operation is repeated multiple times for each tuple of parameters $k \in [1, 30]$ and $threshold\_factor \in [1, 10]$.

The result is visible in figure 27

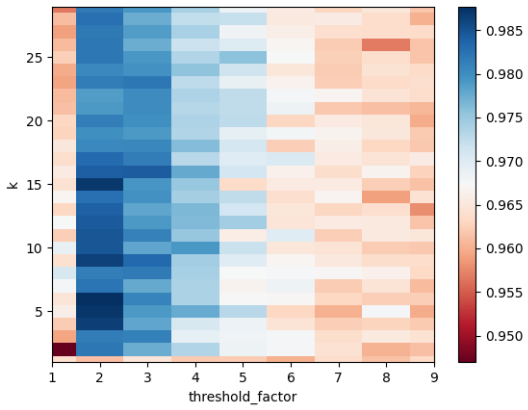

Figure 27: Filtering accuracy in function of k and threshold_factor

[4] It may not be representative of all the scenes the lidar could observe but it aims to describe a common traffic scene. The derived parameters may need fine tuning to fit a different scene.

For the model, the optimal parameters are k=6 and threshold_factor=2. Even though those parameters are optimized for the model and not for the real data, the model is assumed to be good enough to use those parameters, at least as basis with further fine tuning, for the real data. The example of a filtered frame is visible in figure 28

Figure 28: Filtering example of a frame

The top image is the non-filtered frame, while the bottom is the filtered one. We clearly see that the majority of non-targets points are removed. However, the result is not perfect as some non-targets point are still visible. We will deal with these points in the clustering section. Also, some points of the vehicle have been deleted.

6.3.4 Clustering

Clustering in 3D lidar point cloud is still an open research subject as many papers on the subject have been published over the past five years. Also a complete overview of the state of the art is available at [26]. For the related work, one can cite:

- [8] Propose the "Scan Linerun Clustering" which see lidar point cloud as the pixels of an image.
- [14] Propose a clustering algorithm based on the spatial distribution of the point cloud.
- [13] Propose a Window-Based Lidar Clustering (WBLC) for real-time analysis.

The proposed clustering follows the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. DBSCAN is widely used in the literature for clustering data for different applications. We can cite [36], which propose a lane detection scheme from lidar points, based on DBSCAN. More advanced schemes based on a modified version of DBSCAN for lidar point cloud have been proposed in [21] and [16].

The following explanation on DBSCAN is derived from [28]. This algorithm has been particularly chosen for two main advantages. It doesn't need a prior knowledge of the number of

clusters and it can also detect points that belong to noise. As some of those points survived to filtering, it will be useful.

As its name suggests, the algorithm is based on density. It uses two parameters, ϵ and $MinPts$. ϵ is the maximum distance between two points to be considered as neighbor. $MinPts$ is the minimum number of points in a cluster. If this minimum is not reached for a cluster, the related points are excluded as noise. The pseudo-code of the clustering is the following:

```
def extend the clusterof the point:
        if point not yet proceesed:
            nbrslist=find neighbors of the point in a range epsilon
            if not enough points (<MinPts) in neighborhood:
                label the point as noise
            else:
                label the point with its cluster name
                recursively reapply the cluster extension on the neighbor points
        if point already processed and not first point of a new cluster:
            if labeled as noise:
                it is a border point, label by the cluster name


for each point in the poincloud:
    extend the cluster of the point
```

First, all the points are labeled as "not processed" which is label "-2". Then for each point, if this point hasn't already been processed, all the neighbors in a range ϵ are computed. If the point has less than MinPts neighbors, it is classified as noise. Else, it is labeled by the number of the actual cluster. This operation is repeated recursively over all the neighbors and neighbors of neighbors. Finally when a whole cluster is discovered, cluster_label is incremented for the next cluster.

For the $MinPts$, we will use, as for the model in the filtering section, a minimum of 5 points per target. For ϵ, if we choose a too small distance, targets will be split in multiple clusters each. If the distance is too big, multiple targets that are close to each other will be clustered as one target. A trade-off has been made with $\epsilon = 1.5m$. Results are visible in figure 29 where each color represents a different cluster. Here, to better illustrate the clustering, a different scene with two vehicles is proposed.

Figure 29: Clustering of a frame with two vehicles

6.4 Combination scheme

In this section, we will discuss the proposed scheme for combining the modalities. After the processing described in the previous section, for each of the three modalities, the data is formatted as follows:

- **Camera data** For each frame, a list of the targets containing the bounding boxes, the classes and the levels of confidence.
- **Radar data** For each frame, a list of the targets containing distance and speeds.
- **Lidar data** For each frame, a list of the targets containing 3D position of the centers and number of points in the clusters.

The proposed scheme is the following:

- **1.** Associate targets between the three modalities.
- **2.** Track the targets between the frames.

6.4.1 Target associations

As the YOLOv8 is an algorithm that has already been tried and tested in the literature, it is chosen as the basis for the beginning of the associations. We are going to associate to each target detected by YOLOv8, the most probable cluster from the lidar's data, and the most probable maximum in radar's Doppler graphs.

Camera-Lidar: The metric to evaluate the lidar cluster that is most likely the target from the camera depends on two comparisons. First, the difference between the azimuth angle of arrival of the center of the bounding box (camera) and the center of the cluster (lidar). Second, a comparison between the size of the bounding box and the number of points in the cluster. Both the size and number of points decrease with the distance and increase with size of targets.

Lidar-Radar: Points from the Radar are associated to lidar clusters based on the distance returned by both modalities.

6.4.2 Tracking

The tracking is achieved based on "ByteTrack" introduced in 2022 in [22]. It is a framework that proposes a strategy to associate bounding boxes between following frames. The approach is represented in figure 30 from [24].

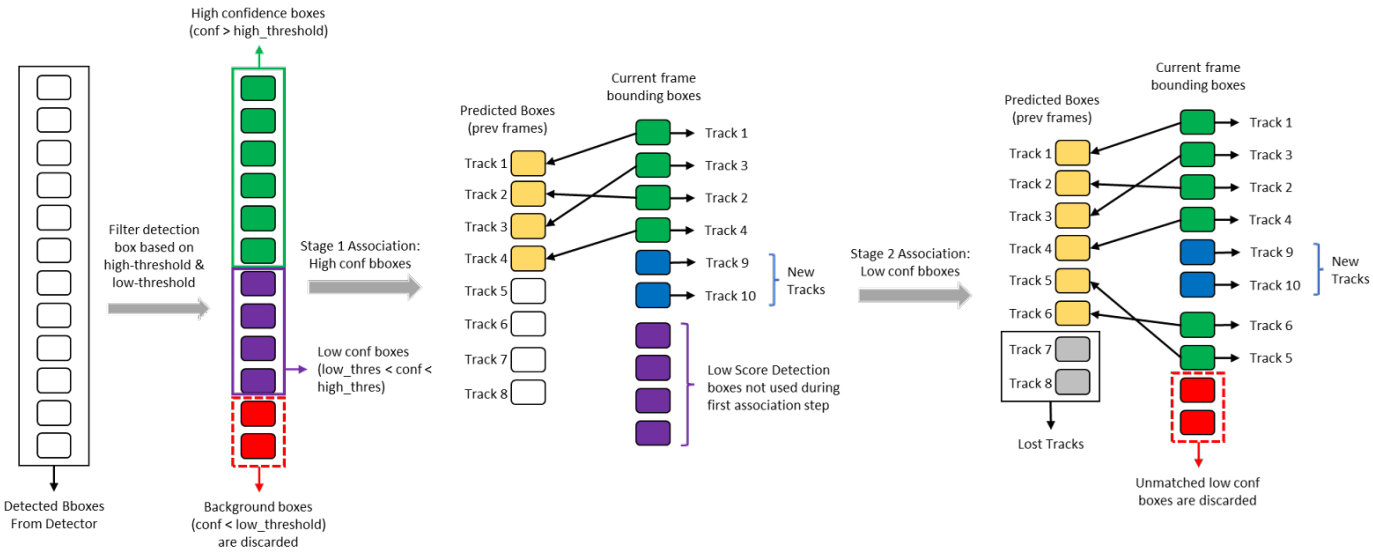

Figure 30: Bytetrack strategy [24]

First bounding boxes are computed normally, then, they are sorted by confidence level and separated in three groups:

- **1) High threshold < Confidence :** It includes all the bounding boxes that would normally be kept as their confidence level is above the threshold.
- **2) Low threshold < Confidence < High threshold:** It includes the bounding boxes that have a confidence level above a new defined "low threshold" but would normally be discarded as their confidence level is under the "high threshold".
- **3) Confidence < Low threshold:** It includes the bounding boxes with confidence level under the "low threshold". These boxes are classified as background and discarded.

When processing a new frame, boxes from group 1 are first associated with boxes from previous frame. Then boxes from group 2 are associated with boxes from previous frame. Finally boxes from group 3 are discarded.

Associations are based on a comparison between the actual bounding boxes and a prediction of these boxes based on a Kalman filter applied on the previous frame. The comparison metric

is called IoU for Intersection over Union. Bounding boxes with highest IoU between frames are most likely to belong to the same target.

Ultralytics, that provides the YOLOv8 implementation also provides a compatible ByteTrack implementation which will be used here.

6.4.3 Results

The result of the complete scheme is visible in figure 31.

Figure 31: Tracking results

In this figure, each color represents the tracking of a target for few minutes of acquisition. One can see that there are some errors in the tracking. After investigating, it appears that the Bytetrack works well, but the associations between the camera bounding boxes and the lidar clusters is sometimes wrong. This part of the scheme still needs improvements.

6.5 Comparison with previous results

This section aims to prove the lidar's utility in the system by reviewing previous results from [18] and [25] and proposing an alternative to their schemes taking advantage of lidar's data.

6.5.1 Speed ambiguity

As explained by Ledent in [18], FMCW radar parameters choice implies a trade-off between precision and max range. The relation between both is the following:

$$|v_{max}| = (N-1)\Delta v \tag{6.1}$$

v_{max} is the maximum speed without ambiguity, N is the size of the FFT (here 256) and Δv is the speed resolution. With the chosen parameters there is an ambiguity on the speed of the target above $100km/h$. With the help of lidar's data this ambiguity can be easily removed. A simple way to accomplish this is to compute a center of mass of each cluster in a lidar's data frame. Then, the evolution of the position of this center of mass gives an estimation of the speed of the target. This is only to remove the ambiguity. Even though one can extract an estimation of the target's speed, the radar is still needed because as the center of mass

is computed from points at the surface of the vehicle, it is prone to high variations and low precision.

6.5.2 Distance ambiguity

Similarly to speed ambiguity, there is a maximum distance that can be measure by the radar without ambiguity. Again, there is a trade-off between resolution and maximum range following:

$$|d_{max}| = (\frac{N}{2} - 1)\Delta d \tag{6.2}$$

The distance of the target is directly available in the lidar's data. Moreover, from specifications, the lidar distance error is less than $2cm$ at 20m while radar precision is fixed at $27.5cm$ from parameters. This time the lidar's data is more precise.

6.5.3 Tracking

Thomas Bolteau proposed a tracking in [25] that leaded to the tracking path visible in figure 32.

Figure 32: Accumulation of tracking from [25]

Even though, the tracking proposed in the previous section still needs improvement, one can see that the lidar brings a strong improvement in the precision of the position of the targets. In figure 31, the tracks fit the road while there are some impossible track points in 32.

7 Dataset creation

In the scope of algorithm development, and more particularly in the case of neural network, training data is acknowledged as crucial. The are multiple datasets available in the literature, some are more general, as the well known ImageNet [2], some are more specific as AnimalKingdom [19] for animal behaviour understanding.

For traffic scenes, the most part of datasets are ADAS-oriented, as sensors are embedded on moving cars. There are plenty of them, one can cite; Kitti [3], nuScenes [12], Eurocity [10], Pixset [15]. Only two propose the three modalities, camera, radar and lidar but not for static scenes. As far as the performed literature review increased my knowledge, no dataset with the three concerned modalities was available for traffic monitoring in a static environment.

The proposed dataset monitors different traffic intersections from a static point of view. The arbitrary choice has been made to include the full raw data in the dataset. However for storage purposes, the data processing proposed in previous sections will be included as scripts to run by the user and not as processed data.

7.1 Setup

The acquisition setup is visible in figure 33.

Figure 33: Acquisition setup

The three modalities are directly embedded inside and on the box. The whole system is autonomous and powered by 11.1 V batteries. No additional computer is needed as everything is managed by the included *Jetson Nano*®.

7.2 Acquisition process

The acquisition process is fully automatized. When the system is switched on, it creates a directory labeled by the time it was switched on and stores continuously the data from the

three modalities during a pre-programmed period of time. Once the acquisition session is finished, the technician only needs to extract the SD-card from the *Jetson Nano*® to get the data. A 2 minute acquisition represent 2.5 Gb of data. Thus, it is necessary to regularly empty the SD-card.

7.3 Data formatting

The dataset is organized as follows:

```
Dataset
├── Date1
├── Date2
├── ...
├── ...
├── DateX
│   ├── Description.txt
│   └── Data
│       ├── jpeg
│       │   ├── timestamp1.jpeg
│       │   ├── ...
│       │   ├── ...
│       │   └── timestampX.jpeg
│       ├── raw
│       │   ├── timestamp1.raw
│       │   ├── ...
│       │   ├── ...
│       │   └── timestampX.raw
│       └── Lidar
│           └── Prise1.rawlidar
└── ProcessingScripts
```

Each acquisition is labeled by the date. For each acquisition, a description file describing the scene is provided. It contains information about the location (GPS coordinates), weather conditions, traffic density, visibility, etc. Then comes the data. It is divided in three directories, one for each modality. First one, jpeg, contains camera data. One jpeg file is provided for each timestamp. Second one, raw, contains radar data. These are binary files containing the raw data signals from the radar, one for each timestamp. Finally, Lidar, contains the lidar data. All the acquisitions are included in a single binary file. This file is a concatenation of 22 bytes points formatted as; 8 bytes: timestamp, 4 bytes: x-coordinate, 4 bytes: y-coordinate, 4 bytes: z-coordinate, 1 byte reflectivity, 1 byte tag.

Finally, Processing_scripts contains all the post processing scripts used to produce the results from previous sections.

8 Further improvements

Even though synchronization is achieved, there is still room for improvement. First, the three modalities use different synchronization schemes. The camera is connected via USB but the radar and lidar synchronization could be standardized to both use gPTP protocol. Moreover, when the synchronization scheme was designed by Ledent, there was no switch in the system. As the switch adds a variable delay in the communication, the radar synchronization could suffer from a higher uncertainty with the actual synchronization scheme. The delay in a typical network switch is few tens of micro-seconds.

This thesis also proposed processing schemes for data from the three modalities. The radar is the limiting element for the frame rate. As explained, there is a trade-off between range and resolution in the choices of the parameter. The use of the lidar to remove ambiguity could also be used to improve radar performance. One could favor velocity resolution rather than range which is better performed by the lidar. In term of radar signals, it means a smaller bandwidth but more ramps per seconds. Lots of perspectives are conceivable. Moreover, the proposed combination scheme combines data after their processing. There is room for a transformer neural network development that could get the best out of the information from the different modalities at the level of raw data. The reflectivity included in the lidar's data is also a source of information that was not used yet.

Finally, the proposed dataset is promising and still growing. A significant improvement could be to develop and provide a full mapping between the data from the different modalities to help for its use. An association between each camera image pixel and a direction (Θ,Φ) in the lidar's data would be a strong help for the combination scheme development.

9 Conclusion

The two proposed contributions were synchronization of the sensor's modalities and creation of a multi-modal dataset. Now that this is achieved, it opens the door for further research. Integrating data from various sensors allows for comprehensive analysis and understanding of the traffic environment. By fusing information from the camera, the radar, and the lidar, researchers can obtain a more detailed and nuanced representation of traffic patterns, road conditions, and safety hazards.

The lidar's integration brings a perspective of accurate and precise tracking. This tracking could be used for real-time traffic flow optimization, adaptive traffic signal control, or improved incident detection and response. The lidar also provides a mapping of the road infrastructure that can be used for accident detection. The raw data from the lidar brings a complete 3D overview of the scene. It allows research for development of deep learning neural network, that could accurately handle raw data from the three modalities. Finally, the lidar's data includes information about an object's shape and size, that can be used to enhance object identification.

I hope this work will help the research by allowing development of further algorithms based on this combined sensor.

References

[1] Hermann Rohling. "Fast segmentation of 3D point clouds: A paradigm on LiDAR data for autonomous vehicle applications". In: *IEEE Transactions on Aerospace and Electronic Systems AES-19.4*. 1983, pp. 608–621. DOI: 10.1109/TAES.1983.309350.

[2] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[4] R. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *IEEE Conf. Comput. Vis. Pattern Recognit.* (2014), pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[5] W. Liu et al. "SSD: Single shot multibox detector". In: *Computer Vision (Lecture Notes in Computer Science)* 9905 (2016), pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

[6] J. Redmon et al. "You only look once: Unified, real-time object detection". In: *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)* (2016), pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[7] S. Ren et al. "IEEE Trans. Pattern Anal. Mach. Intell." In: *IEEE Conf. Comput. Vis. Pattern Recognit.* 39 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[8] Dimitris Zermas, Izzat Izzat, and Nikolaos Papanikolopoulos. "Fast segmentation of 3D point clouds: A paradigm on LiDAR data for autonomous vehicle applications". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5067–5073. DOI: 10.1109/ICRA.2017.7989591.

[9] Kentaro Yoshioka et al. "A 20ch TDC/ADC hybrid SoC for 240×96-pixel 10%-reflection <0.125%-precision 200m-range imaging LiDAR with smart accumulation technique". In: *2018 IEEE International Solid-State Circuits Conference - (ISSCC)*. 2018, pp. 92–94. DOI: 10.1109/ISSCC.2018.8310199.

[10] Markus Braun et al. "EuroCity Persons: A Novel Benchmark for Person Detection in Traffic Scenes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8 (2019), pp. 1844–1861. DOI: 10.1109/TPAMI.2019.2897684.

[11] Alexis Duflot. *Conception d'un senseur intégré multimodal pour l'observation des routes*. 2019.

[12] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11618–11628. DOI: 10.1109/CVPR42600.2020.01164.

[13] Micaela Verucchi et al. "Real-Time clustering and LiDAR-camera fusion on embedded platforms for self-driving cars". In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. 2020, pp. 398–405. DOI: 10.1109/IRC.2020.00068.

[14] Li C. Gao F. Han X. Zhang B. "A New Density-Based Clustering Method Considering Spatial Distribution of Lidar Point Cloud for Object Detection of Autonomous Driving". In: (2021). DOI: 10.3390/electronics10162005.

[15] Jean–Luc Déziel et al. "PixSet: An Opportunity for 3D Computer Vision to Go Beyond Point Clouds With a Full-Waveform LiDAR Dataset". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. 2021, pp. 2987–2993. DOI: 10.1109/ITSC48978.2021.9565047.

[16] Feng Gao, Caihong Li, and Bowen Zhang. "A Dynamic Clustering Algorithm for Lidar Obstacle Detection of Autonomous Driving System". In: *IEEE Sensors Journal* 21.22 (2021), pp. 25922–25930. DOI: 10.1109/JSEN.2021.3118365.

[17] Kévin De Sousa Gauthier Rotsart de Hertaing. *Conception d'un senseur intégré multimodal pour l'observation des routes*. 2021.

[18] François Ledent. *Synchronization of Multimodal Data Flows for Real-Time AI Analysis*. 2022.

[19] Xun Long Ng et al. "Animal Kingdom: A Large and Diverse Dataset for Animal Behavior Understanding". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 19001–19012. DOI: 10.1109/CVPR52688.2022.01844.

[20] Tolga Turay and Tanya Vladimirova. "Toward Performing Image Classification and Object Detection With Convolutional Neural Networks in Autonomous Driving Systems: A Survey". In: *IEEE Access* 10 (2022), pp. 14076–14119. DOI: 10.1109/ACCESS.2022.3147495.

[21] Mohammad El Yabroudi et al. "Adaptive DBSCAN LiDAR Point Cloud Clustering For Autonomous Driving Applications". In: *2022 IEEE International Conference on Electro Information Technology (eIT)*. 2022, pp. 221–224. DOI: 10.1109/eIT53891.2022.9814025.

[22] Yifu Zhang et al. *ByteTrack: Multi-Object Tracking by Associating Every Detection Box*. 2022. arXiv: 2110.06864 [cs.CV].

[23] Oct. 2, 2023. URL: http://www.webcamerausb.com/elp-2mp-1080p-cmos-ar0330-h264-mjpeg-yuy2-30fps-uvc-usb20-android-linux-windows-driverless-embedded-camera-module-mic-p-89.html.

[24] Dec. 26, 2023. URL: https://www.datature.io/blog/introduction-to-bytetrack-multi-object-tracking-by-associating-every-detection-box.

[25] Thomas Bolteau. *Design and implementation of a multimodal dataset processing pipeline for vehicle tracking model*. 2023.

[26] Adnan M Slavic G Martin Gomez D Marcenaro L Regazzoni C. "Systematic and Comprehensive Review of Clustering and Multi-Target Tracking Techniques for LiDAR Point Clouds in Autonomous Driving Applications." In: *Sensors* (2023). DOI: 10.3390/s23136119.

[27] *Communication protocol of Livox Hap*. Nov. 20, 2023. URL: https://github.com/Livox-SDK/Livox-SDK2/wiki/Livox-SDK-Communication-Protocol%20HAP(English)#23-Data-Types.

[28] *DBSCAN*. Dec. 20, 2023. URL: https://en.wikipedia.org/wiki/DBSCAN.

[29] *gPTP synchronization explanations*. Dec. 1, 2023. URL: https://inet.omnetpp.org/docs/showcases/tsn/timesynchronization/gptp/doc/index.html.

[30] *gPTP synchronization from Livox*. Dec. 1, 2023. URL: https://livox-wiki-en.readthedocs.io/en/latest/tutorials/new_product/common/time_sync.html#gptp-time-synchronization.

[31] *Iowait explanations*. Oct. 25, 2023. URL: https://haydenjames.io/what-is-iowait-and-linux-performance/.

[32] *Ip protection rating*. Dec. 14, 2023. URL: https://fr.wikipedia.org/wiki/Indice_de_protection#cite_note-cc-5.

[33] *Laser safety*. Oct. 6, 2023. URL: https://en.wikipedia.org/wiki/Laser_safety.

[34] *Livox user manual*. Dec. 14, 2023. URL: https://terra-1-g.djicdn.com/65c028cd298f4669a7f0e40e50b
Livox%20HAP%20(TX)%20User%20Manual.pdf.

[35] *Livox website*. Oct. 6, 2023. URL: https://www.livoxtech.com/hap.

[36] B Roja Reddy et al. "Implementation of Lane Detection Algorithms for Autonomous Vehicle Using Lidar Point Cloud Data". In: *2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*. 2023, pp. 1–6. DOI: 10.1109/CSITSS60515.2023.10334206.

[37] *RFbeam website*. Oct. 5, 2023. URL: https://rfbeam.ch/product/k-md2-engineering-sample/.

[38] *Switch reference*. Dec. 18, 2023. URL: https://www.ldlc.com/fr-be/fiche/PB00143085.html.

[39] *Time documentation*. Nov. 23, 2023. URL: https://www.gnu.org/software/libc/manual/html_node/Getting-the-Time.html.

[40] *Unix clock reference*. Oct. 18, 2023. URL: https://en.wikipedia.org/wiki/Unix_time.

[41] *Walloon roads safety research*. Dec. 28, 2023. URL: https://infrastructures.wallonie.be/home/nos-thematiques/routes/securite-routiere/amenagements/equipements-de-securite.html#.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**