UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

University of Padova

DEPARTMENT OF INFORMATION ENGINEERING
Master's Thesis in ICT for Internet and Multimedia

Design and Evaluation of Machine Learning algorithms to support Predictive Quality of Service in Vehicular Networks

Supervisor:
Prof. Marco Giordani

Co-supervisor:
Dr. Paolo Testolina

Candidate:
Filippo Bragato

Academic Year 2022/2023

Alle persone, vicine o lontane, che in questi anni
hanno costruito la grande famiglia, fatta di affetto,
cura, passione e rispetto, che siamo noi.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

6G	Sixth Generation
AD	Autonomous Driving
AI	Artificial Intelligence
AP	Average Precision
APP	Application
D-DQN	Double Deep Q-Learning Network
DNN	Deep Neural Network
e2e	end-to-end
FoV	Field of View
gNB	Next Generation Node Base
GT	Guida Teleoperata
ICP	Iterative Closest Point
iid	independent and identical distributed
IoU	Intersection over Union
KPI	Key Performance Indicator

LiDAR	Light Detection and Ranging
LOS	Line of Sight
mAP	Mean Average Precision
MCS	Modulation and Coding Scheme
MDP	Markov Decision Process
ML	Machine Learning
OFDM	Orthogonal Frequency Division Multiplexing
PDCP	Packet Data Convergence Protocol
PHY	Physical
PQoS	Predictive Quality of Service
PQS	Predizione della Qualità di Servizio
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RL	Reinforcement Learning
RLC	Radio Link Control
SELMA	SEmantic Large-scale Multimodal Acquisitions
SGD	Stochastic Gradient Descent
SINR	Signal to Interference plus Noise Ratio
TD	Teleoperated Driving
UE	User Equipment

Abstract

This thesis focuses on Teleoperated Driving (TD) applications to enhance road safety, where vehicles equipped with sensors are remotely controlled by teleoperators using real-time data delivered by the network. To perform TD safely, the network has to satisfy several Key Performance Indicators (KPIs), especially in terms of delay and reliability, which is not trivial due to the variability of the network conditions.

To solve this issue, the research community is investigating Predictive Quality of Service (PQoS) to estimate network conditions and perform adequate countermeasures in case communication KPIs are not satisfied.

Motivated by the potential of Artificial Intelligence (AI) in PQoS, this thesis proposes several solutions based on Machine Learning (ML) to perform PQoS. First, given the importance of data to train ML algorithms, we worked on an extension of SELMA, a new multimodal synthetic dataset for autonomous driving, that we modified to provide multiple correlated series of samples acquired by sensors moving in an urban environment.

Then, we present two different paradigms to perform PQoS. The first is based on goal-oriented communication, and exploits the degree of correlation between point clouds to evaluate when automotive data should be sent. The second investigates the trade-off between Quality of Service (QoS) and Quality of Experience (QoE) in the attempt to identify the optimal level of compression for automotive data. Specifically, the extended SELMA dataset has been used to train and test our PQoS frameworks, and simulation results show that the correlation between point clouds can be approximated onboard with an energy-efficient metric based on the symmetric point-to-point Chamfer Distance, and that a Reinforcement Learning (RL) agent can effectively take countermeasure to address communication KPIs while maximizing the QoE.

Sommario

In questa tesi viene proposto uno studio per migliorare la sicurezza della Guida Teleoperata (GT). Nella GT i veicoli sono controllati a distanza da un teleoperatore, grazie alla trasmissione in tempo reale dei dati raccolti dai sensori del veicolo. Per garantire la sicurezza nella GT, è necessario che la comunicazione rispetti una serie di requisiti di affidabilità e latenza, operazione tuttavia complicata dalla variabilità delle condizioni di rete e di canale.

Per questo la comunità scientifica sta studiando il paradigma della Predizione della Qualità di Servizio (PQS) quale strumento in grado di predirre le condizioni di rete e di modificare i parametri di comunicazione per rispettare i vincoli di sicurezza.

Visto il successo dell'uso dell"intelligenza artificiale nelle tecniche di PQS, in questa tesi vengono proposte diverse soluzioni basate sull'apprendimento automatico per la PQS.

In questo contesto, vist l'esigenza di dati per l'allenamento di algoritmi di intelligenza artificiale, come prima cosa in questa tesi viene proposta un'estensione del dataset artificiale SELMA. Nell'estensione vengono infatti resi disponibili dati provenienti da acquisizioni sequenziali di sensori a bordo di veicoli che si muovono in uno scenario realistico.

Inoltre vengono proposti due diversi paradigmi con cui fare PQS. Il primo, basato sulla "comunicazione orientata agli obiettivi", sfrutta la correlazione tra acquisizioni consecutive di uno stesso sensore per calcolare la rilevanza della nuova acquisizione; il secondo, basato sul bilanciamento tra compressione e accuratezza nell'operazione di rilevamento di oggetti, mira a calcolare il livello di compressione più adeguato alle condizioni di canale percepite dal veicolo.

L'estensione di SELMA è stata usata per allenare e validare i paradigmi proposti, ed i risultati empirici dimostrano come sia possibile approssimare la rilevanza dei dati acquisiti a bordo del veicolo tramite algoritmi che non richiedono un'eccessiva potenza computazionale, e che sia possibile trovare il livello di compressione ottimale per la trasmissione dei dati utilizzando un agente di apprendimento per rinforzo.

Chapter 1

Introduction

1.1 Teleoperated Driving

In the Teleoperated Driving (TD) setup, vehicles equipped with sensors are controlled remotely by an agent called a teleoperator, which relies on the network to receive almost real-time data from the vehicle and react accordingly [1]. It is important to notice that the teleoperator can be a human or a sufficiently powerful machine. Depending on the teleoperator, TD can be exploited in different situations.

With a human teleoperator, TD is considered a key element for supporting autonomous vehicles when the automation fails, letting the teleoperator control the vehicle whenever the automation system stops working [2]. If a system/mechanical failure occurs, the automated system will halt the vehicle at a secure location and establish a connection with an operator center. An operator at the center can then investigate the cause of the failure and, if required, assume control of the automated vehicle. This approach offers the benefit of requiring fewer operators compared to the number of automated cars, thus removing the necessity for a safety driver within each vehicle [3].

On the other side, when the objective is a working vehicle without an onboard driver, TD is performed with a server acting as a teleoperator. This approach is cheaper than in the case of Autonomous Driving (AD), and requires less onboard computational power.

One of the main problems regarding studying TD is the availability of data. Especially, data are crucial to develop research and perform realistic simulations, for example to train and test Machine Learning (ML) algorithms that are required for TD. The most important examples are object detectors, i.e., ML algorithms that are able to identify and locate objects in samples gathered from sensors onboard the vehicles; notably, the accuracy of the object detection operation is fundamental when performing TD with a machine as a teleoperator.

Therefore, a lot of effort has been spent in trying to obtain automotive datasets from fully simulated data. In particular in [4] researchers proposed a fully-simulated dataset based on CARLA, an open-source simulator to support the development, training, and validation of autonomous driving systems.

Finally, researchers highlighted the feasibility of TD, but there are lots of critical issues regarding the feasibility of the communication [5], in particular due to the variance present at the Radio Access Network (RAN). This concern is justified by the setup of TD: if we imagine transmitting a raw high-resolution Light Detection and Ranging (LiDAR) perception every 0.1 s, we need an approximate average uplink data rate of 250 Mbps. This data rate can be difficult to handle in resource-constrained networks especially considering the tight network requirements for TD.

To address this concern, it is crucial to take steps that prevent changes in network conditions from causing failures in the TD application. In this context, Predictive Quality of Service (PQoS) has emerged as a viable solution [6].

1.2 Predictive Quality of Service

Autonomous systems should always estimate how the environment in which they operate will evolve to ensure that network Key Performance Indicators (KPIs), e.g., in terms of throughput, delay, and reliability, are continuously satisfied. On the other side, the network is known to be prone to high variability, making the task of predicting the behavior of the network challenging. The authors in [6] proposed a possible solution to this problem: PQoS. In detail, PQoS is a set of techniques that aims at predictively inferring the future condition of the network, and to act accordingly in order to always respect the KPIs.

Notably, PQoS is considered a fundamental tool for the realization of use cases such as AD and TD [7]. This is motivated by the fact that in an urban scenario, due to the mobility of User Equipments (UEs), i.e., the vehicles, network conditions suffer from an even higher variability, and due to the safety constraints of those applications, the respect of the KPIs is fundamental to avoid any risk of injuries.

To date, there have been several attempts to model the Quality of Service (QoS) from a statistical point of view [8], e.g., using game theory. But, compared to traditional techniques, ML and Artificial Intelligence (AI) have been proven to be more effective in their ability to precisely predict the QoS in short time intervals [9]. However, the ultimate purpose of PQoS is not only to predict network changes and/or fluctuations but also to react accordingly; in this sense, due to the stochastic nature of the environment in which vehicles move and operate, appropriate countermeasures at the network levels have to be made to make it sure that network requirements are satisfied, As such, Reinforcement Learning (RL) is clearly amongst the most suitable solutions to address the issue of PQoS. Several attempts have been made to use RL [10] and Deep RL [11] to perform PQoS. Still, in the TD scenario, most of literature work contains too many unfeasible and/or simplistic assumptions to be deployed in a real system. For example in [12] and [10] the proposed solutions are based on the segmentation of point clouds at the transmitter, but segmenting point clouds requires more computational power than performing object detection, and therefore is not achievable at the UE.

1.3 Goal-Oriented Communication

Since the definition of the theory of goal-oriented communication [13] there has been a growing interest in this topic. This theory proposes a shift in the telecommunication framework. In particular, communication is no longer intended as a simple exchange of information but rather as a collective action to reach a specific goal.

Thanks to its potential, this theory is expected to play a central role in Sixth Generation (6G) communication networks. For example, the authors in [14] promote the idea of going beyond the Shannon paradigm, focusing only on the reception of the relevant information.

Given the growing interest in this perspective, researchers have started developing new goal-oriented sampling and communication policies in the field of autonomous systems [15]. In these regards, several theoretical analyses have been made to prove the effectiveness of goal-oriented communication via Edge Learning [16] and for resource allocation [17].

Given the attention drawn by goal-oriented communication and the proven effectiveness of this framework, we believe that a goal-oriented approach could significantly improve the performances of PQoS. However, in the context of TD, a definition of how information can be marked as "relevant" is yet missing. Such a definition is not trivial due to the different types of acquisitions and data that are used in TD. Therefore, in this thesis, a new definition of "similarity" based on correlation between subsequent LiDAR acquisitions is proposed, with the objective of computing the value of information of point clouds.

The value of information is expected to play a central role in the transmission of sensors' data [18]. In particular, the availability of value of information in point clouds makes it possible to perform informed decisions on the transmission and priority of data. Still, to this aim, value of the information must be known at the transmitter and, due to the limited computational power and resources available onboard the vehicles, this is not trivial. For this reason, we proposed several metrics to approximate the value of information in a TD scenario, which do not require hardware-expensive operations to be performed onboard the vehicles.

1.4 Thesis Objectives and Structure

Based on the above introduction, the main objective of this thesis is to propose and validate an end-to-end (e2e) framework for performing PQoS in a TD scenario. The main difference with respect to previous research is the realism of our approach, meaning that this work will take into account constraints on the UEs and the RAN that have been previously ignored, specifically the computational capacity of end users.

First, an analysis of the degree of similarity between subsequent acquisitions of LiDAR data is proposed, and several measures are identified to approximate this similarity. This is done to compute onboard the value of the information of new point clouds, thus supporting the principle of goal-oriented communication, and avoiding the transmission of redundant data. Second, an analysis of the effects of the compression of point clouds is presented. This enables the training of an RL agent to perform PQoS based on the trade-off between QoS and Quality of Experience (QoE). The RL agent is trained and tested to find the optimal compression mode for point clouds, and simulation results in ns-3 ensure the realism of the outcomes. We claim that the proposed framework is able to help the vehicles to choose carefully the relevant data to send based on the scope of TD, thus achieving goal-oriented communication. The ultimate goal is to reduce the burden of data transmission through the network, which may improve communication performance.

Since both approaches require the availability of data for training, in this thesis we used and extended the SEmantic Large-scale Multimodal Acquisitions (SELMA) dataset for autonomous driving research. SELMA [19] is a new open-source synthetic dataset recently released by the University of Padova, and is developed to mimic as close as possible a real TD environment. Specifically, SELMA is a realistic dataset that contains synthetic sensors' acquisitions from vehicles in an urban scenario. SELMA consists of data from multiple and different sensors, town models, weather and traffic conditions, and acquisitions from different times of the day. As such, thanks to its versatility, it stands out as an optimal tool for TD research.

SELMA is built on top of the CARLA simulator. In particular, in Chapter 2 we developed a new technique to control the behavior of pedestrians, avoiding problems with traffic management. Other precautions are taken to regulate the behavior and placement of vehicles on the map so to have realistic and meaningful mobility patterns. Finally, a new management system for the bounding boxes has been implemented in SELMA in order to ensure better performance for object detection.

Then, SELMA is used to study the degree of correlation between point clouds. In Chapter 3 we provide a theoretical definition of correlation between point clouds in the context of TD. Then, we propose several techniques to approximate the correlation between point clouds that can be performed onboard the vehicles and do not require particularly expensive and/or power-consuming computer vision operations. The first technique is based on the correlation of the voxel representations of point clouds. It is really fast but, due to the nature of point clouds and the specifics of an urban environment, it does not scale well as the number of points in the background increases. The second one is based on the clusterization of points in the point clouds using an adapted version of the Lloyd's k-means algorithm. However, the presence of outliers in the background also introduces noise and artifacts in this measure. Finally, the last technique is based on a modified version of the Chamfer Distance, and was proven to be an accurate metric to measure the correlation of point clouds compared to its competitors.

In Chapter 4 we present a complete e2e framework for PQoS. Specifically, point clouds are compressed onboard using Draco, an open-source program developed by Google. The compression level is chosen by an RL agent that implements the Double Deep Q-Learning Network (D-DQN) algorithm. The agent is trained with the network as the environment and with rewards that are proportional to the KPIs of the application. Notably, the D-DQN is trained via federated learning, which is a promising approach in 6G. Finally, the compressed point cloud is sent to an edge server, and it is processed to detect mobile elements in the scene. The detection is done using PointPillars, a common object detector for point clouds. In this sense, the quality of the detection is measured in terms of the Mean Average Precision (mAP), which is used to compute the reward and train the D-DQN.

The D-DQN algorithm is evaluated via simulations in ns-3 in Chapter 5. Numerical results show that the D-DQN algorithm is able to finely balance the compression level to maximize the mAP while satisfying most of the KPIs of the TD application. However, performance degrades quite quickly as the number of vehicles in the network increases, due to the fact that transmission resources may not be enough to support data transmissions of point clouds, although after compression.

To solve this issue, future research should investigate an approach to optimize both the degree of correlation of automotive data (so as to estimate the level of "new" information in the point clouds, and decide when data should be sent) and the relative level of compression to minimize the size of data, as discussed in the conclusions in Chapter 6.

Chapter 2

SELMA

The core of this thesis is finding new techniques and new approaches to perform TD. The main problem regarding TD is the heavy burden it leaves on the network. In particular, the transmission of big unordered structures can lead to network congestion, and, due to the tight requirements in terms of KPIs that this application has, congesting the network is not affordable.

Most of the proposed solutions in this thesis are data-driven, meaning that they required data to work, but also in the sense that data were needed to validate or fugate some hypotheses.

The dataset needed for this thesis must fulfill some requirements. The most important requirement is realism, meaning that the dataset must contain samples that reflect a real driving environment, indeed the dataset must contain all elements that could reasonably be in an urban environment, and all the mobile elements must move in the city with realistic behavior. This is crucial to avoid artifacts that could invalidate the proposed solution in a real environment.

Another fundamental requirement of the dataset is correctness, meaning that points in the point clouds must be correctly segmented and the position of the objects in the samples must be known. Furthermore, the dataset must contain correlated samples, meaning that the dataset must be divided into scenes and each scene must contain subsequent acquisitions of the same sensor in a dynamic environment.

Finally, to better capture the correlation between subsequent acquisitions, a high sampling rate is needed.

The last three requirements, given the number of expected frames and the cost of manually labeling a sample, can not be satisfied by an empirical dataset, therefore the dataset must be a synthetic dataset.

Unfortunately, none of the existing synthetic datasets fulfills the requirements for this thesis [20]. And for this reason, the thesis required the creation of a new synthetic dataset adequate to the TD scenario, the SELMA dataset.

The simulator chosen to create the SELMA dataset is CARLA [21]. The CARLA simulator has been employed as a pivotal tool in the creation of an artificial dataset for TD and AD research. As an open-source simulation platform, CARLA provides a controlled virtual environment that replicates real-world driving scenarios, enabling the generation of data resembling authentic driving conditions. This dataset serves as a valuable resource for assessing and training TD and AD algorithms. The controlled nature of CARLA's simulation environment allows for the systematic collection of data under various scenarios, offering researchers the ability to explore specific aspects of TD and AD performance. By utilizing CARLA to craft this artificial dataset, the complexities of real-world driving are encapsulated, contributing to the evaluation and refinement of TD and AD systems.

The CARLA simulator is written in C++ and it is based on the UnrealEngine, but exposes Python APIs to ease the process of creating simulations. From now on the notation based on the Python APIs will be used to refer to elements inside CARLA. The convention used to refer to classes is `carla.Class` while for methods is `carla.Class.method(args)`. To improve the readability, when the class is clear from the context, methods are referred to as `method(args)`.

The creation of high-quality datasets is a critical aspect of advancing research in the field of TD and AD. In this chapter, the intricate process of generating the SELMA dataset is delved into, focusing on the challenges encountered and the innovative solutions employed to address them. SELMA is a comprehensive dataset designed to facilitate research and development in the realms of TD and AD. This chapter sheds light on the significant steps taken to refine the dataset's realism, diversity, and usability, with a particular emphasis on tackling issues related to pedestrians, bounding boxes, and vehicle management.

Generating an artificial dataset that accurately mimics real-world scenarios using the CARLA simulator is a task full of challenges. One of the most prominent obstacles encountered was the inclusion of pedestrians in the dataset. Since pedestrians constitute a crucial element of road dynamics and safety, their representation is an essential requirement. However, CARLA's inherent handling of pedestrians presented significant missing pieces.

The management of pedestrians within CARLA proved to be far from straightforward. CARLA's built-in class, `carla.WalkerAIControl`, intended for controlling pedestrian movement, has a behavior that is not suitable for the creation of this dataset. In particular, it often led pedestrians to cause problems with their behavior, resulting in both vehicular accidents involving pedestrians and traffic congestion due to cars waiting for walkers.

In response to the challenges posed by CARLA's pedestrian management, a novel solution was introduced: the `SmartWalker` controller. Unlike `carla.WalkerAIControl`, which exhibited suboptimal path selection, the `SmartWalker` controller was designed to address these issues comprehensively. The core concept behind the `SmartWalker` controller lies in its ability to understand the proximity of pedestrians to the street and to enable them to walk without interfering with the traffic flow.

Another critical aspect of the dataset is the accurate representation of bounding boxes for vehicles. While CARLA provides APIs for bounding box manipulation, issues were encountered related to the representation of two-wheeled vehicles and pedestrians. The dataset's focus on object detection necessitated precise bounding box information. By manually adjusting bounding box parameters and leveraging saved information, these challenges were overcome and more accurate bounding box representations were achieved.

The realism extends beyond pedestrian behavior to encompass the overall dynamics of the road environment. Issues related to vehicle management, particularly in preventing the destruction of actors due to accidents or traffic congestion, are addressed. Adjusting traffic light timings and vehicle spawn locations, along with introducing randomness to vehicle positioning and behavior, contributed to a more realistic and diverse road environment.

Furthermore, the dataset was enriched by introducing Smart Vehicles equipped with an array of sensors, including cameras and lidars. This addition enhances the dataset's suitability for exploring cooperative perception, where data from multiple sensors are merged to improve environmental understanding. The placement of these sensors on the vehicles is discussed, and their role in enhancing the dataset's utility for research purposes is highlighted.

The culmination of these efforts resulted in the creation of the SELMA dataset, a comprehensive resource tailored to TD and AD research. The dataset's diverse features, encompassing multiple towns, varying traffic densities, different times of day, and various weather conditions, make it an invaluable tool for researchers and practitioners. With its emphasis on realism, diversity, and the intricacies of pedestrian-vehicle interactions, SELMA offers a unique platform for advancing the state of the art in autonomous driving.

2.1 Walkers

This is not the first attempt to create a synthetic dataset using CARLA, but the obtained results are inadequate to fulfill the requirements of this thesis, the main problem is the lack of realism of most datasets based on CARLA, due to the way in which walkers are implemented in CARLA.

Obtaining realistic data using the CARLA simulator has proven to be challenging due to the way in which CARLA manages pedestrians, the control of pedestrians, indeed, is not straightforward. For this reason, some datasets avoid entirely the problem, not including walkers [22], others instead do not allow for vehicles to move due to the high number of walkers and lack of walkers' control [23].

Since pedestrians are the most vulnerable users of the street and each AD or TD solution must ensure their safety, including them in the SELMA dataset was one of the main requirements.

From the implementation point of view, it is important to understand why the management of walkers is not trivial in CARLA. In fact, CARLA itself provides a class `carla.WalkerAIControl`, that takes care of the movement of pedestrians, but it suffers several issues.

Using the CARLA notation, `carla.WalkerAIControl` is an invisible Actor attached to a Walker, which is itself an Actor.
The `carla.WalkerAIControl` is not rendered in the simulation by any means and it is only responsible for the movement of Actors. The interactions between `carla.Walker` and `carla.WalkerAIControl` are shown in Fig. 2.1.

In particular, through the method `go_to_location(destination)` it is possible to set the new destination of the walker and let the controller take care of the movement of the Actor.

Even though in these terms the process seems trivial, there are several issues to overcome.

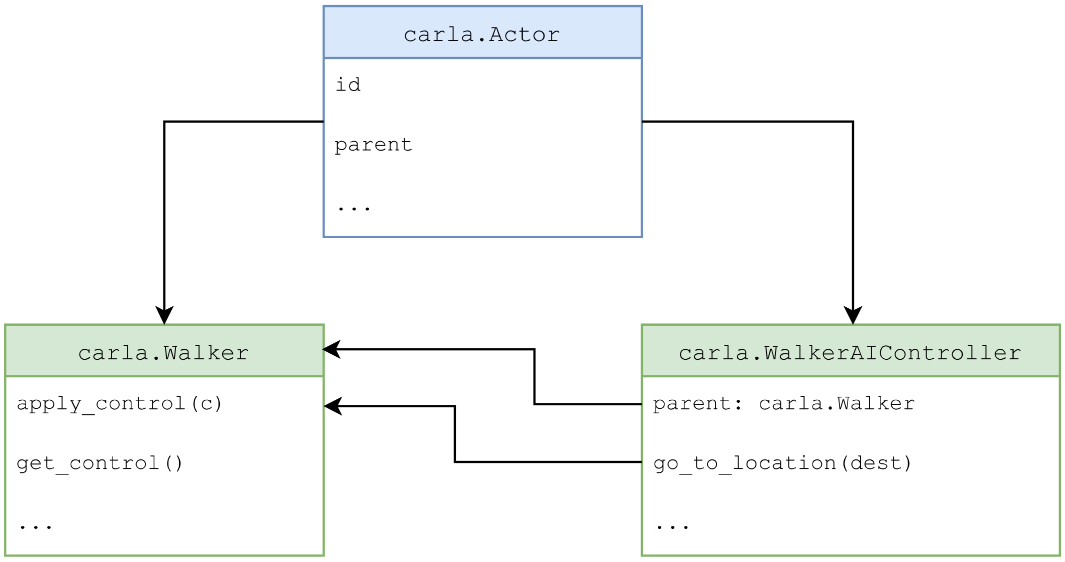


Figure 2.1: Class diagram highlighting the relationship between walker and controller

The main problem is related to how the `carla.WalkerAIControl` chooses the path that the walker must follow.

In particular, the controller often makes the pedestrian cross the street even when the destination is on the same sidewalk as the walker. This leads to several problems for the simulation.

Since CARLA's map are relatively small and with a relatively high number of junctions, cars waiting for people to cross the road create strong congestion in the road network, that propagate around the map until any car is barely moving.

Another problem with the `carla.WalkerAIControl` is that it does not take care of the distance between the walker and the street, meaning that the controller does allow pedestrians to walk really close to the street. But, the probability of ignoring pedestrians must be set to 0%, otherwise, the simulation will inevitably experience a pedestrian hit by a car, and when the probability is set to 0%, several issues in the mobility of the cars arise due to the presence of pedestrians close to the street.

When the probability is set to 0% a car, seeing that a person is too close to the street, will slow down and eventually stop even if the pedestrian is not walking towards the street, creating the same congestion problem as before.

So one of the main things to take care of when designing the behavior of pedestrians is that they must walk at least at some distance to the street, without this requirement the simulation will not be realistic.

For this reason, the use `carla.WalkerAIControl` was avoided, and a controller for the walkers able to deal with all those problems was implemented.

2.1.1 Custom Walkers Controller

Following the rationale of CARLA, a class to deal with this task was created, the name of the class is `SmartWalker`, and as `carla.WalkerAIControl`, each instance of `SmartWalker` must be linked to a Walker.

As stated before one of the most important pieces of information to consider is the distance between walkers and the street. This information was modeled as a vector $\mathbf{d} \in \mathbb{R}^8$, whose entries are the Euclidean distance between the walkers and the closest point of the street in specific directions.

To formalize the definition of the vector $\mathbf{d}$, the function $c(\mathbf{w}, \theta)$ is defined. c is a function that takes in input $\mathbf{w}$, a position on the projection of the map in a 2D plane, and θ, an angle measured clockwise with respect to the north, and gives the position $c(\mathbf{w}, \theta)$ of the closest point of the road in the direction θ, starting form position $\mathbf{w}$ as stated in Eq. 2.1.

$$\begin{aligned} c : \mathbb{R}^2 \times [0, 2\pi] &\to \mathbb{R}^2 \\ \mathbf{w}, \theta &\mapsto c(\mathbf{w}, \theta) \end{aligned} \tag{2.1}$$

Given the c function, being $\mathbf{w}$ the position of the walker, the i-th entry of the vector $\mathbf{d}$ is defined as follows:

$$\mathbf{d}[i] = \|\mathbf{w}, c(\mathbf{w}, i\pi/4)\|_2, \tag{2.2}$$

so **d** contains information of the distance in directions spaced by 45 degree. Experimentally, It can be seen that this level of resolution is enough given the fact that towns in CARLA have perpendicular streets and sidewalks that follow cardinal directions. A representation of **d** is shown in 2.2.

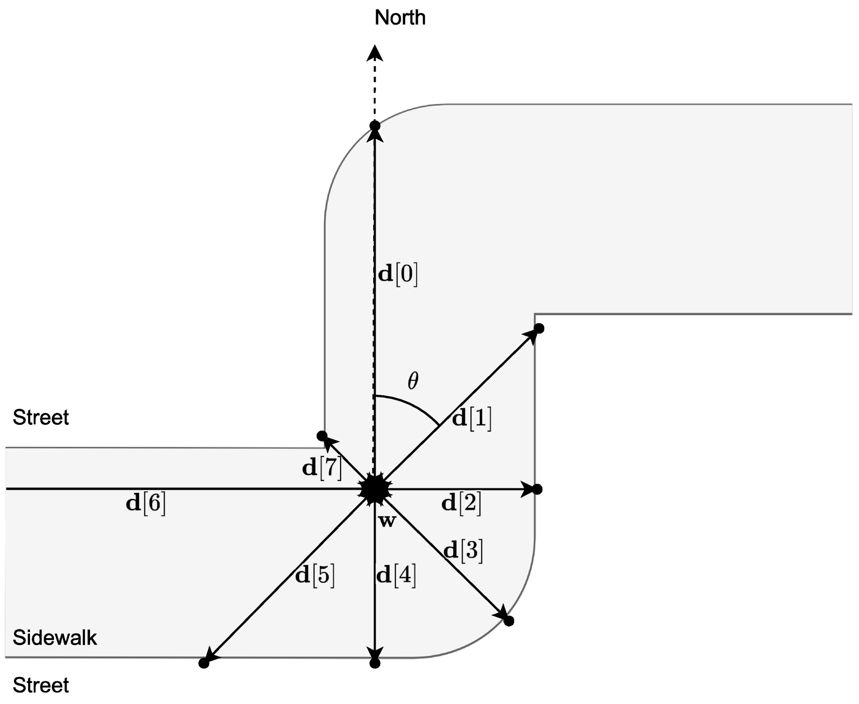


Figure 2.2: Representation of the vector **d**

The vector **d** is used for taking decisions on the direction towards which the walker has to move.

Each instance of `SmartWalker` is then defined by a state s, an integer that represents the status of the walker (e.g.: they are going far from the street, they are walking towards his next waypoint, they have arrived at his waypoint, and so on).

Finally, each `SmartWalker` has a variable that represents the direction θ_w that the walker is following, which has the same convention as the angle in the c function.

The behavior of the waker can be summarized with the following rules:

- At the beginning of the simulation the walker is spawned at the center of the sidewalk and is assigned to a random direction $\theta_w = \mathcal{U}(0, 2\pi)$.

- Walks towards θ_w for ρ meters.

- If the distance between the walker and his next waypoint is less than the threshold α, the walker has arrived at its destination.

  When it arrives, the `SmartWalker` will choose the next direction, being θ_w the actual direction and $j : \theta_w = j\pi/4$, the direction chosen is

  $$\theta_w = \frac{\pi}{4} \cdot (\arg\max_k \{\mathbf{b}_{k \bmod 8} | k \in \{(j-2), j-1, \ldots, j+2\}\}). \qquad (2.3)$$

- If at any time, any entry of $\mathbf{d}$ is less than σ, the walkers turn to the opposite direction. Being $\mathbf{d}_k < \sigma$, the walker will move towards $\theta_w = (4+k)\pi/4$ for ρ meters.

- If at any time the distance between the walker and its next waypoint is non-decreasing with respect to the distance measured in the previous time step, the walker is considered unable to move and walks toward $\theta_w + \pi$.

For the simulations, I used the parameters described in Table 2.1.

Symbol	**Description**	**Value**
α	Distance to see if walker has arrived	0.5m
ρ	Distance to differentiate walkers	1.0m
σ	Distance to see if walker is close to the street	0.5m

Table 2.1: Parameters used for the simulator

2.1.2 Map Management

`carla.Map.get_waypoint` was used to compute $c(\mathbf{w}, \theta)$, `get_waypoint` is a method that returns the type of lane at a specific `carla.Location`.

This is once again a problem because the vector $\mathbf{d}$ must be computed at each time step for each walker, and to compute $\mathbf{d}$ the function `get_waypoint` must be evaluated several times.

CARLA's calls on the `carla.Map` object are heavily time-consuming, and the straightforward implementation of the described procedure required approximately three weeks for 120 seconds of simulation time.

Since that was not affordable, the computation of $c(\mathbf{w}, \theta)$ must be implemented in a smarter way. The idea was to create a data structure containing an approximate value of $c(\mathbf{w}, \theta)$ for each $\mathbf{w}$ on the sidewalk and for each θ in $[0, \pi/4, \ldots, 7\pi/4]$.

First `get_waypoint` was used to sample the map in a regular grid. Then a bidimensional array with boolean entries was created, the array contains `True` if and only if the sampled point associated with the entry is a sidewalk.

This data structure contains all the information needed to compute $c(\mathbf{w}, \theta)$ on each point of the grid. For convenience, let $\hat{c}(\mathbf{w}, \theta)$ be the function that approximates $c(\mathbf{w}, \theta)$ considering only the points on the grid.

Also in this case, the naive algorithm reported in Alg. 1 was too slow to compute $\hat{c}(\mathbf{w}, \theta)$ in a reasonable time.

With a grid of size $n \times n$ the Algorithm 1 takes $O(n^3)$ operation, because we have $O(n^2)$ points on the grid, and, for each point, the most inner loop is repeated $O(n)$ times.

The idea to solve it using dynamic programming is that all consecutive points on the sidewalk along a direction share the same $\hat{c}(\mathbf{w}, \theta)$.

For example, being $\mathbf{w}$ a point on the sidewalk with a known $\hat{c}(\mathbf{w}, \theta)$, we know that if the closest point to $\mathbf{w}$ on direction $\theta - \pi$ (called $\mathbf{w}'$) is on a sidewalk, it is ensured that $\hat{c}(\mathbf{w}', \theta) = \hat{c}(\mathbf{w}, \theta)$.

Algorithm 1: NaiveImplementation(m)

```
Data: m: structure containing true if the point is on a sidewalk
Result: $\hat{c}(\mathbf{w},\theta)$ approximating $c(\mathbf{w},\theta)$
$\hat{c}(\mathbf{w},\theta) = \texttt{Null} \quad \forall\theta \in [0, \pi/4, \ldots, 7\pi/4], \quad \forall\mathbf{w}$ on sidewalks;
for $\theta \in [0, \pi/4, \ldots, 7\pi/4]$ do
    for $\boldsymbol{w}$ on grid do
        $\mathbf{s} \leftarrow \mathbf{w}$;
        while $m[\boldsymbol{s}]$ do
            $\mathbf{s} \leftarrow$ closest point to $\mathbf{s}$ on the grid on direction $\theta$;
        end
        $\hat{c}(\mathbf{w},\theta) \leftarrow \mathbf{s}$;
    end
end
```

Thanks to this, we can write a dynamic program version of the naive algorithm, reported in Alg. 2 and Alg. 3.

Algorithm 2: cStarter(m)

```
Data: m: structure containing true if the point is on a sidewalk
Result: $\hat{c}(\mathbf{w},\theta)$ approximating $c(\mathbf{w},\theta)$
$\hat{c}(\mathbf{w},\theta) = \texttt{Null} \quad \forall\theta \in [0, \pi/4, \ldots, 7\pi/4], \quad \forall\mathbf{w}$ on sidewalks;
for $\theta \in [0, \pi/4, \ldots, 7\pi/4]$ do
    for $\boldsymbol{w}$ on grid's border do
        $\hat{c}(\mathbf{w},\theta) \leftarrow$ cFinder($m, \hat{c}(\cdot,\cdot), \mathbf{w}, \theta$);
    end
end
```

Here is important to notice that Alg. 2 calls $O(n)$ times Alg. 3, and Alg. 3 calls itself $O(n)$ times before returning to Alg. 2.

This leads to the fact that the complexity of this solution is $O(n^2)$, and since we have $O(n^2)$ points to analyze this is the best result we can achieve in terms of asymptotic complexity.

By storing the result of this procedure in a proper data structure and by letting the `SmartWalker` controller access this data structure I was able to significatively speed up the time needed for the computation and realize a dataset containing realistic walkers.

Algorithm 3: cFinder($m, \hat{c}(\cdot,\cdot), \mathbf{w}, \theta$)

```
Data: m: structure containing true if the point is on a sidewalk,
$\hat{c}(\cdot,\cdot)$: structure storing the results,
$\mathbf{w}$: the position of the walker,
$\theta$: the direction toward what we are looking to.
Result: $\hat{c}(\mathbf{w}, \theta)$
$\mathbf{s} \leftarrow$ closest point to $\mathbf{s}$ on the grid on direction $\theta$;
if $m[\boldsymbol{s}]$ then
    return cFinder($m, \hat{c}(\cdot,\cdot), \mathbf{s}, \theta$)
else
    return $\mathbf{s}$
end
```

2.2 Bounding Box

Another major problem with the data generated using CARLA is that they are mainly thought to perform point cloud segmentation. In fact, thanks to the possibility of programmatically labeling the point clouds while generating them, CARLA is particularly suitable for this type of dataset.

But for the scope of this thesis, I needed to perform object detection on the point clouds, and for this objective having segmented point clouds is not enough.

Also in this case CARLA exposes some APIs to work with the bounding boxes of Actors in the Map, but unfortunately, the native API suffers from several issues.

The first and most serious problem was related to bikes and motorbikes. In particular, the CARLA management of two-weels vehicles is completely different from the management of cars.

Each Actor instance in the CARLA engine has a location. In the case of cars, the location is the point on the plane where the car touches the ground, in the middle of the car (at half the length and width of the vehicle).

Each Actor contains an instance of the class `BoundingBox`, which is characterized by a location and an extent. The location is the offset of the location of the center bounding box with respect to the location of the Actor. This value is expressed in the coordinate system of the vehicle, and so the rotation of the vehicle does not affect this value. The extent, instead, is half of the length of the side of the bounding box.

Two-weel vehicles, instead, have the location of the actor in the back right corner of the vehicle, their associated bounding boxes do not compensate in the x-y plane for this, and they usually have the extent equal to 0 in different dimensions.

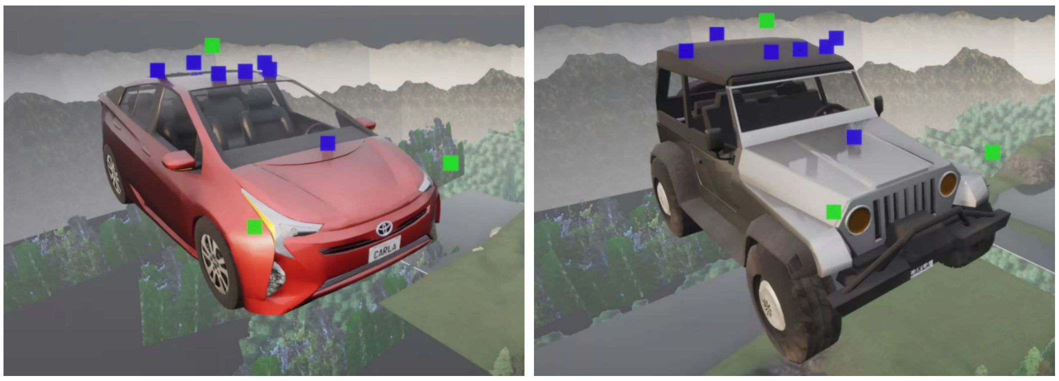
For this reason, bounding boxes of motorcycles are usually wrong and create a lot of trouble when reconstructing the scene. To avoid this an empirical approach has been used.

The first step was to save all the correct information of bounding boxes (namely the extent and the location) in a JSON file. To gather this information the debugger available in CARLA was used and values of the extent and the location were manually tweaked.

Then, when saving the features of the bounding box during the simulation, the values in the actors were ignored, and instead, the values in the JSON file were used.

Another similar problem was related to pedestrians. In particular, the extent of the bounding boxes of most pedestrians was the same, even thou pedestrian models have significantly different sizes.

For this reason, most of the points in LiDAR acquisitions were outside of the bounding boxes. In this case, the the same techniques as the motorcycles were used, and, after creating the suitable bounding box parameters for each pedestrian, the problem was solved.



(a) Toyota Prius (b) Jeep Wrangler Rubicon

Figure 2.3: Some examples of the positioning of sensors in Smart Vehicles, blue dots represent cameras, while green dots represent LiDARs

2.3 Vehicle Management

To refine the dataset, some adjustments to the way vehicles are managed were made.

The main problem was with the fact that vehicles that can not properly move are removed from the scene, and this is not acceptable in a realistic dataset.

The main source of the destruction of Actors is the involvement in a car accident. In most cases, the `TrafficManager` of CARLA is enough to prevent car accidents, but in some cases, more attention is required, especially when spawning vehicles.

The first thing is to avoid spawning vehicles on junctions because, in that case, the behavior of the Traffic Manager is unpredictable. Furthermore, if cars are spawned too close to a traffic light, they will ignore it at the beginning of the simulation, causing road accidents.

Another situation leading to the destruction of a vehicle happens when one still vehicle is in the way of another vehicle. This can happen consistently on short roads that end with a traffic light.

To address this issue, the timing of traffic lights in the simulations was adjusted, and the number of spawned vehicles was modified, interpreting this occurrence as a clear indicator of city congestion.

For the purpose of obtaining more realistic data, the average distance between each vehicle and the one ahead was also modified to match real-world conditions. Additionally, the vehicles were programmed to randomly deviate from the center of the lane, replicating human driving behavior.

Finally, 15% of the cars in the simulation are equipped with sensors, We refer to them as Smart Vehicles. Each Smart Vehicle has 7 cameras and 3 lidars. Cameras are placed to cover all the surroundings of the vehicle. One LiDARs is placed on top of the car (with a Field of View (FoV) of 360 degrees), and the other two are placed near the headlights of the car and have a FoV of 270 degrees. Some examples are shown in Fig. 2.3.

2.4 Results

Feature	**Description**
Towns	`Town01_Opt`, `Town02_Opt`, `Town03_Opt`
Smart Vehicles	The dataset contains sensors' data for over 200 vehicles
Sensors	For each Smart Vehicle there are 7 RGB cameras, 7 depth cameras, 7 segmentation cameras, and 3 lidars
Traffic	`None`, `Low`, `Medium`, `High`
Daytime	`Noon`, `Sunset`, `Night`
Weather	The dataset contains samples of 7 different weather conditions

Table 2.2: SELMA's features

As reported in Tab. 2.2, SELMA has a strong variability under several aspects. The most relevant for this thesis is the number of different vehicle data we can analyze.

Furthermore SELMA is a great dataset for exploring AD and TD using simulation, and the publication of this dataset could fuel the research on those topics. In particular, thanks to the availability of data from different vehicles (Fig. 2.4), SELMA enables the studying of cooperative perception, which consists of merging data from different sensors to improve the understanding of the environment and the scene [24].

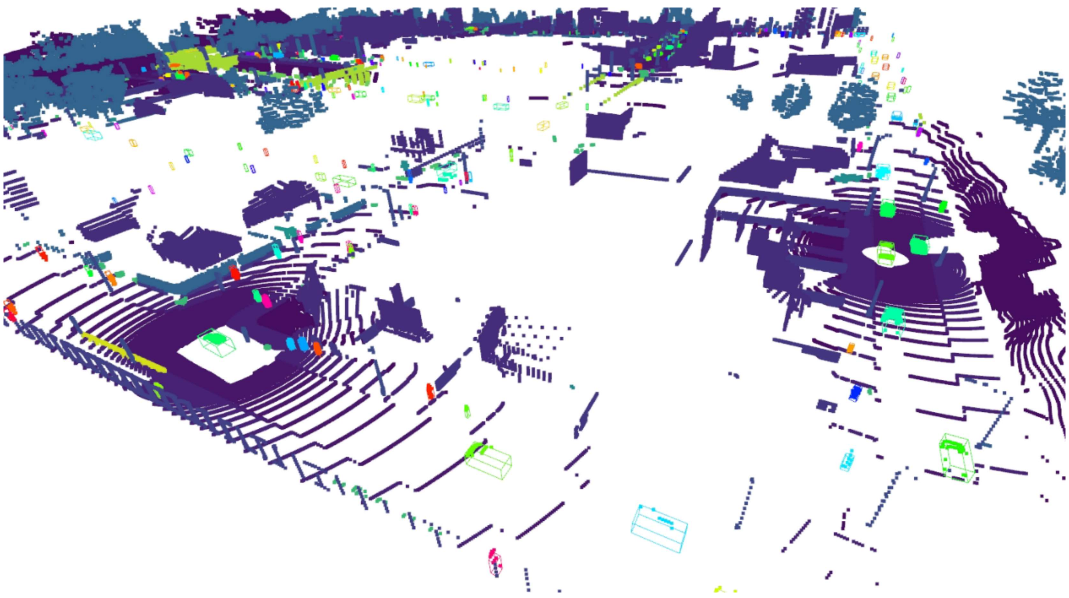


Figure 2.4: Example of SELMA's acquisitions

Chapter 3

Point Clouds Analysis

One of the main requirements for 6G mobile systems, is the shifting towards a decentralized orchestration. The network is expected to support 10^7 users for each km^2 in order to grant a pervasive connection. But to coordinate the communication among all those devices, end terminals need to make autonomous network decisions, avoiding a central orchestrator [25].

Indeed, 6G will let end users autonomously decide the numerology of the transmission, adapting on their own to the status of the network, according to a decentralized paradigm [26].

In the context of TD, it is easy to see that one of the biggest advantages that vehicles have for taking this type of decision is the availability of data. In particular, they can inspect and process data before sending them, taking advantage of the information they can retrieve to make informed decisions on communication.

For instance, considering point clouds, vehicles have the possibility to choose the compression level before sending data, or if the point cloud is considered not enough informative they could also decide to discard it and analyze the next one.

This perspective left some open questions. In particular, due to the specifications of teleoperated driving, we expect vehicles with limited computational power to impose harsh limitations on the type of preprocessing that can be done at the transmitter.

Ideally, knowing the position of every object in the surrounding of the vehicle will lead to the most informed decision, but performing object detection is considered too computationally demanding for the vehicle. For this reason, the choice of the operations to perform on the point cloud is not clear yet.

Following this rationale, the core idea of this chapter is to analyze different types of metrics to try to infer the quantity of new information present in the point cloud with respect to previous acquisitions, limiting the computational power needed to compute this value.

In order to formalize this question we need to formally define our target. The aim is to develop a fast algorithm to quantify the dissimilarity between point clouds. Calling $\mathbf{P}_k$ the point cloud acquired at time k, we define the dissimilarity as $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$.

To the best of our knowledge, a proper definition of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ is not present in the literature, due to the fact that this measure is context-dependent. Therefore a definition of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ will be presented in Sec. 3.0.1, while in Sec. 3.1 will be proposed some metrics to approximate the dissimilarity defined in Sec. 3.0.1, that are computationally compliant with the capacity of vehicles.

3.0.1 Dissimilarity Definition

Given the unordered nature of point clouds, having a coherent dissimilarity measure based on the point cloud itself is almost impossible, and could lead to undesired results.

For example, a LiDAR mounted on the top of a vehicle, during each acquisition frame, registers a large number of points in close proximity to the car. Those points will be mainly related to the vehicle itself or the street. When the vehicle moves, the points of the vehicle and the street will change at each acquisition, due to the movement of the sensor. This brings big changes in the point cloud. Those changes are only linked to the information that the car is moving, and this information is known to the receiver. Meaning that this huge change in the point cloud does not have a semantic meaning for the receiver.

Another important thing to consider is that most of the points in the point clouds are related to static objects, such as the street, buildings, or other elements in the city. Consecutive point clouds may differ greatly, as they capture the same static objects from different points of view.

This explains why the mere position of points is not enough informative when measuring the dissimilarity between point clouds, and more meaningful metrics must be found processing those points.

In the context of TD, the main interest is being able to identify the exact location of mobile objects in the scene, since we can assume that the receiver has a complete tridimensional map of the static elements of the environment.

It is important for the dissimilarity measure to be able to capture this information. For this reason, the proposed solution to compute the dissimilarity is based on the positions of mobile elements in the scenes.

We denote the set of mobile elements visible from the lidar at time t as $\mathcal{M}_t$. An object m is considered visible from the LiDAR if the object is in Line of Sight (LOS) of the LiDAR and the point cloud contains a minimum of v points belonging to object m. Subsequently, the function $p_t(m)$ is introduced, which associates each object m with its position at time t within the coordinate system of the LiDAR.

Given two time instants t and $t + n$ and their related set $\mathcal{M}_t$ and $\mathcal{M}_{t+n}$ the union of those sets $\mathcal{M}_t \cup \mathcal{M}_{t+n}$ can be partitioned into three sets:

- $\mathcal{M}_t \cap \mathcal{M}_{t+n}$ represents all mobile objects that are present in both acquisitions. Indeed, it represents the information shared between different acquisitions.

- $\mathcal{M}_t \backslash \mathcal{M}_{t+n}$ represents all mobile objects that are present at time t but not at time $t+n$, that are object exiting the range of the LiDAR, or objects not visible anymore. This represents information that was present at time t, but not at time $t+n$, i.e., it is outdated information.

- $\mathcal{M}_{t+n} \setminus \mathcal{M}_t$ represents all mobile objects that are present at time $t+n$ but not at time t. This is the "new" information, the one that was not available at time t, but at time $t+n$ is available.

Furthermore, it is important to notice that objects in $\mathcal{M}_t \cap \mathcal{M}_{t+n}$ might be in completely different positions in the two time instants. This leads to the necessity of including this information in δ.

Also the position of objects in $\mathcal{M}_{t+n} \setminus \mathcal{M}_t$ is crucial because having really far new objects in LOS is not as relevant as having new close objects. In the context of TD, indeed, an unseen element that appears in close proximity to the vehicle can cause damage to the vehicle and the object itself.

Therefore, to sum up all those considerations, the following formula, based on the Euclidean distance $\|\cdot\|_2$, is proposed to compute the dissimilarity. In this formula $\mathcal{M}_k$ is the set of visible elements at time k, $p_k(m)$ is the position at time k of the object m and α is the parameter that controls how close new objects must be to be considered new important elements

$$\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = \sum_{m \in \mathcal{M}_t \cap \mathcal{M}_{t+n}} \|p_{t+n}(m) - p_t(m)\|_2 + \sum_{m \in \mathcal{M}_{t+n} \setminus \mathcal{M}_t} \frac{\alpha}{\|p_{t+n}(m)\|_2}, \quad (3.1)$$

Some real examples of the evolution of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$, obtained with the SELMA dataset, are reported in Fig. 3.1. Each line reports the evolution of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ as the lag n between point clouds increases, with different levels of traffic. In particular, this plot was obtained with $v = 1$ and $\alpha = 10^4$.

Some general consideration on $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ can be inferred from the information present in Fig. 3.1.

- The dissimilarity between a point cloud and itself is 0. This is easily

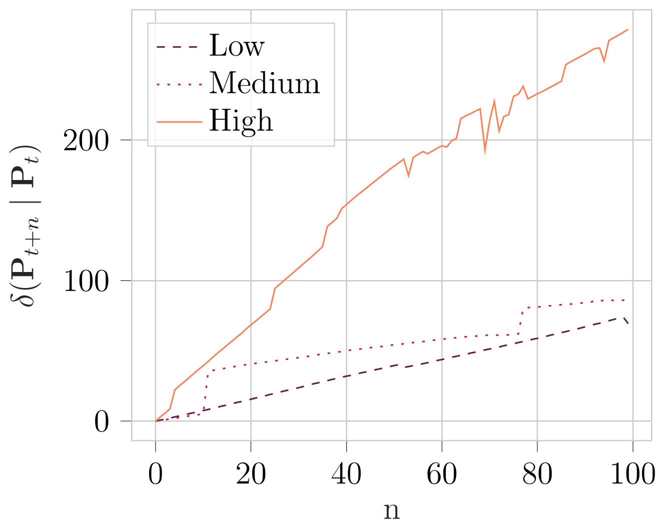


Figure 3.1: Evolution of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ as the lag n increases, with $\upsilon = 1$ and $\alpha = 10^4$, varying the traffic level

verifiable from the formula, since the $\mathcal{M}$ sets are the same, and the positions of mobile elements are the same. $\delta(\mathbf{P}_t \mid \mathbf{P}_t) = 0 \quad \forall \mathbf{P}_t$.

- The number of mobile elements influences the measure in the simulation. Indeed, δ grows faster for higher traffic scenarios. Even if this is generally true, there are some situations where this might not be exact (let's think, for example, of situations where there are lots of vehicles stopped at a traffic light).

- δ is generally non-decreasing with respect to n, but also in this case there are situations in which this is not true. For example when the cardinality of $\mathcal{M}_t$, $|\mathcal{M}_t| = 0$, $|\mathcal{M}_{t+1}| = 1$ and $|\mathcal{M}_{t+2}| = 0$,

$$\frac{\alpha}{\|p_{t+1}(m)\|} = \delta(\mathbf{P}_{t+1} \mid \mathbf{P}_t) > \delta(\mathbf{P}_{t+2} \mid \mathbf{P}_t) = 0. \tag{3.2}$$

But, on the transmitter side, the computation of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ requires an onboard object detector and tracker, which is not feasible due to their high computational demands. As a result, the necessity arises to identify rapid algorithms that can approximate δ.

In Sec. 3.1 several methods to approximate δ are defined and analyzed. This process is done assuming the position and orientation of the sensor known at each time step. This is justified by the fact that the vehicle is equipped with a precise GPS, but it can be relaxed. Indeed the movement of the vehicle can be inferred by performing registration on the point clouds, which means aligning the point clouds until points are superposed with each other. Also in this case we have lots of possible implementations, and with a simple Iterative Closest Point (ICP) technique, we could remove this assumption [27].

3.1 Different Metrics for the Dissimilarity

3.1.1 Voxelization

The first algorithm proposed is directly linked to correlation. For this reason, it is not a dissimilarity measure, but a similarity one.

The similarity σ can be thought of as the reciprocal of the dissimilarity δ

$$\sigma(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = \frac{1}{\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)}. \tag{3.3}$$

Since the dissimilarity exhibits linear behavior, the behavior of the similarity is hyperbolic, as shown in Fig. 3.2.

The standard definition for the correlation between two discrete signals U, V in $[0, T]$ is the following:

$$r(U,V) = \frac{\sum\limits_{t\in[0,T]} (U[t]-\bar{U})(V[t]-\bar{V})}{\sqrt{\sum\limits_{t\in[0,T]} (U[t]-\bar{U})^2 \sum\limits_{t\in[0,T]} (V[t]-\bar{V})^2}}. \tag{3.4}$$

And even though the extension for tridimensional signals in $\mathbb{R}^3$ is trivial, the problem is that point clouds are not structured and the formula can not

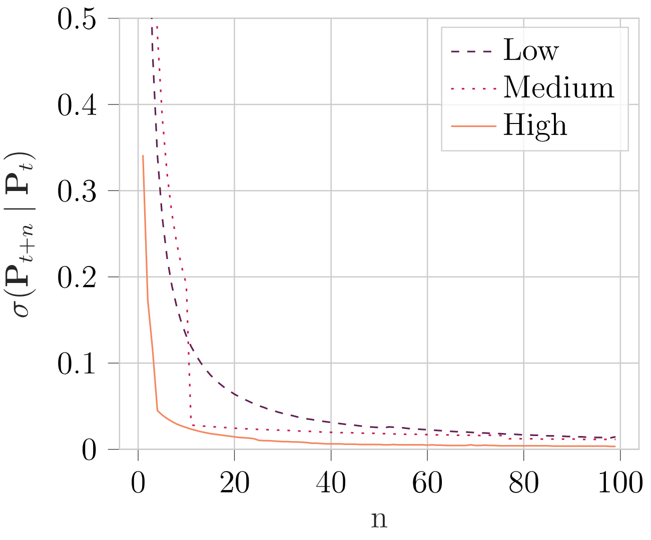


Figure 3.2: $\sigma(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ varying traffic density

be applied directly.

To stick with this correlation definition it is necessary to sample the point clouds in the tree dimension and quantize them. This process is known as voxelization.

The voxel representation of point clouds, or of 3D objects, is well-known in computer vision. It is based on the same principle of pixels where the signal of the image is approximated with a bidimensional matrix. The position of the pixel is encoded in its column and row index, while the value itself contains information about the light intensity of the point.

In the voxel case, the structure is a tridimensional matrix, with each dimension of the matrix representing a direction in $\mathbb{R}^3$, and the entry itself represents information about the points inside the volume covered by that voxel.

The key parameter for voxelizing a point cloud is the resolution. As for images, we can make each voxel represent an arbitrary amount of volume, and by changing the resolution of the representation we get different results.

In SELMA dataset, the maximum dimension of point clouds is (200 m $\times$ 200 m $\times$ 50 m). Voxelizing them with a resolution of 20 m, meaning that each voxel represents a cube with side 20 m, means losing almost all the information present in the point cloud. On the other side, with a resolution of 1 mm, we get a structure with $2 \cdot 10^{15}$ voxels, that is simply not tractable.

The other main feature of the voxelization is the meaning of the entries of the tridimensional matrix. For grey images, the intensity level of light is used. For point clouds, different choices can be made, two of which are presented in this paragraph. The first and most simple is a boolean structure, that has an entry equal to true if and only if there is at least one point inside the volume associated with each voxel. On the other side, we can express in the entry of the voxel the number of points in the point cloud associated with that voxel.

The first implementation is referred to as "Simple" and the second one as "Cumulative", and the associated functions that transform point clouds in voxel representations $v_s(\mathbf{P})$ and $v_c(\mathbf{P})$ respectively.

Two possible ways to compute the similarity between point clouds based on those functions can now be defined, that are respectively:

$$\sigma_s(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = r(v_s(\mathbf{P}_t), v_s(\mathbf{P}_{t+n})) \tag{3.5}$$

and

$$\sigma_c(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = r(v_c(\mathbf{P}_t), v_c(\mathbf{P}_{t+n})). \tag{3.6}$$

However, to assess their performance, it is necessary to visualize their behavior using the same set of point clouds employed in Fig. 3.2, to see whether σ_s and σ_c are good approximations of σ. The obtained results for both σ_s and σ_c concerning their variations with respect to n are presented in Fig. 3.3.

Upon observation, the anticipated hyperbolic pattern is evident, and the correlation seems to exhibit minimal sensitivity to the chosen voxelization type. Henceforth, the choice is made to employ the "Simple" voxelization method, since it is faster to compute and due to the boolean structure is also less computationally demanding.

However, the behavior of the δ curve corresponding to the High traffic

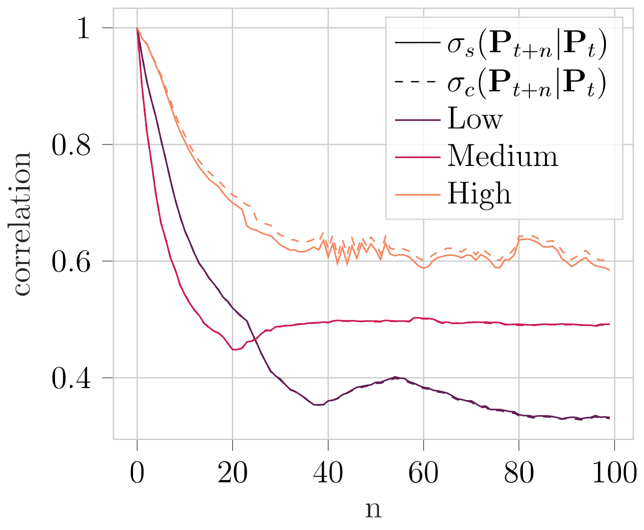


Figure 3.3: Correlation computed using voxels

condition reveals a steeper inclination. Ideally, this would be accompanied by a lower σ hyperbolic curve. Contrary to expectations, the plot suggests that the newly introduced samples in this series hold comparatively lower values.

This is because, due to the high traffic, the car remains still. For this reason, the points related to the landscape are not changing throughout different acquisitions. Furthermore, since most voxels contain only the landscape and not mobile elements, we can say that σ_s and σ_c are not approximations for σ, which is instead only based on the position of mobile elements.

There are other operations that can be easily performed to improve those results.

First, changing the dimension of voxels might give better results. In particular, the plot in Fig. 3.3 is generated with cubic voxels of side 50 cm. As already mentioned, changing the dimension of the voxels correspond to changing the resolution. Specifically, a higher resolution can better capture small changes in the mobile object, while a lower resolution can overcome the noise produced by small changes in the points related to the background due to the evolution of the FoV.

Furthermore, a large number of points belongs to the street. The relative position of the LiDAR with respect to the car p_l is known and constant. In most cases, the car is almost perfectly aligned in the x-y plane with the street; therefore, points related to the street can be removed before the voxelization. This operation simply consists of removing all the points whose z-coordinate is below $-p_l$ before aligning the point cloud with the reference system of the world.

To be more robust and to remove also sidewalks and other static elements all points below $-p_l + \varepsilon$ are removed. This procedure was checked visually ensuring that in this way almost every point related to the street is deleted, losing a negligible fraction of points related to mobile elements.

The pruning function is denoted as Π and is defined as follows:

$$\Pi(\mathbf{P}) = \{p \text{ if } p_z > -p_l + \varepsilon \quad \forall p \in \mathbf{P}\}. \tag{3.7}$$

Therefore, also the definition of similarity changes:

$$\sigma_\Pi(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = r(v_s(\Pi(\mathbf{P}_t)), v_s(\Pi(\mathbf{P}_{t+n}))). \tag{3.8}$$

The value of the dimension of voxels vs is omitted from the formula for clarity.

The results for $\sigma_\Pi(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$, computed on the same data as the previous plot, are reported in Fig. 3.4.

It is clear that $\sigma_\Pi(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ is not a good approximation of σ. The main problem is with the fact that the correlation of the point clouds in the scenario with high traffic is higher than the correlation with fewer vehicles.

As expected different values of vs produce completely different results. Indeed, the higher vs, the higher the correlation due to the lack of resolution. But also in this case the outcome is too dependent on the points of static objects. For this reason, the correlation computed on the voxels is not a suitable choice for this procedure.

Trying again to use the voxels to compute this metric the background

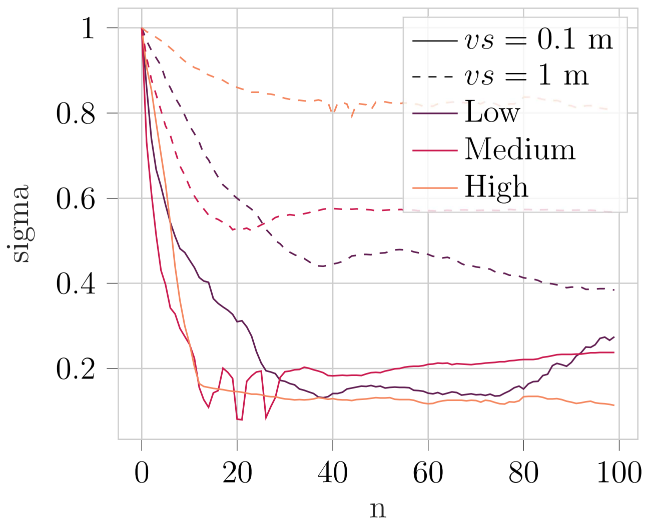


Figure 3.4: Correlation computed using different voxel sizes

should be removed completely. To do so, the presence of a complete tridimensional map of the area must be ensured at the transmitter, and this requirement is not so easy to satisfy.

Furthermore also removing all the points of the background, the quantization introduced by the voxels could lead to undesired behavior. Indeed the movement of a vehicle of the first vs will lead to a drastic decrease in the correlation, while the following movement will not be so effective in changing the correlation.

Thus, in this context, voxelization is not a proper way to compute the similarity between different point clouds.

3.1.2 Clusterization

The second approach taken into analysis is based on clusterization. The presence of an element on the street changes the density of acquired points. Indeed each element, still or mobile, when acquired by a LiDAR, creates a volume in the point cloud where points are more densely distributed. This dense region is the zone corresponding to the surface of the object facing the LiDAR.

On the other side, this also creates a shadow, a region where there are no

LiDAR points due to the fact that the LOS is blocked by the object.

This behavior can be exploited to compute an approximation of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$. The core idea is to cluster the point clouds and use the clusters to compute the dissimilarity between point clouds.

The principle of using clusters to reduce the dimensionality of a dataset is really well known in the literature [28]. Recalling the definition of clusters, elements belonging to the same cluster are similar to each other, while elements belonging to different clusters significantly differ from each other.

Thanks to this feature, all the elements of a cluster can be approximated with a single element that represents the average behavior of all the points inside the cluster.

In the context of point clouds, the coordinates of the points themselves serve as their defining features. This configuration is widely recognized, and numerous studies have been conducted on this subject [29]. Therefore, a brief overview of the key concepts pertinent to this study will be provided.

To begin, it is essential to establish a measure of similarity or dissimilarity among points within the point clouds. Given their characterization in $\mathbb{R}^3$, the distance metric is a suitable choice for dissimilarity. Specifically, the focus will be on the L_2-distance, or Euclidean distance, to facilitate the creation of clusters containing closely located points.

The subsequent consideration is the identification of the most representative element within each cluster. Given that the clusters are formed based on the Euclidean distance, a logical choice is to select the point that minimizes the sum of distances to all other points within the same cluster. This point, denoted as μ_C, optimally satisfies the following minimization criterion for cluster C:

$$\sum_{c \in C} |\mu_C - c|_2. \tag{3.9}$$

This point, referred to as the centroid of the cluster, is straightforward to calculate. Notably, it can be demonstrated that the coordinates of the centroid correspond to the arithmetic mean of the coordinates of all points encompassed within the cluster.

This identified point, referred to as the centroid of the cluster, can be easily calculated. It is noteworthy that the coordinates of this centroid align with the arithmetic mean of the coordinates of all the encompassed points within the cluster.

$$\mu_c = \frac{\sum\limits_{c \in C} c}{|C|}. \tag{3.10}$$

While performing this type of analysis, some researches introduce a further constraint, that is the belonging of the centroid to the cluster. In this case, computing μ_C requires an additional step, and it can be found as:

$$\mu_c = \arg\min_{\mu_C \in C} \sum_{c \in C} \|\mu_C, c\|_2. \tag{3.11}$$

This is done to ensure the fact that the most representative point of the cluster belongs to the cluster itself. But this might lead to suboptimal results in the context of point clouds.

The objective of this clusterization is, indeed, to partition the point cloud into clusters representing objects, taking advantage of the inhomogeneity introduced by the presence of objects. LiDAR points only represent positions on the surface of the object. If we think of the entire object and we want to approximate it with a point the natural idea is the center of mass, which is not on the surface of the object.

This is the intuitive explanation of why the first definition of the centroid is more suitable to represent clusters.

From a technical point of view, the point cloud is partitioned using Lloyd's k-means algorithm [30]. The choice of this algorithm is related to the fact that it lets the user choose a priori the number of clusters, and this is crucial when comparing clusters from different point clouds.

Lloyd's k-means algorithm is conceptually really simple, given the set of points **P** and the number of clusters k, at the beginning k centroids are initialized, then two steps are repeated until convergence is reached:

- Each point is assigned to the closest centroid.
- The position of each centroid is assigned to the mean of the coordinates of all the points inside the cluster.

This procedure is known to be generally really fast, and therefore particularly suitable for this use case. The pseudocode of the algorithm is reported in Alg. 4.

Algorithm 4: Lloyd(**P**, k)

```
Data: P: the set of points
k: the number of clusters
Result: P is partitioned in k clusters
$\mu_1, \ldots \mu_k$ centroids are initialized;
while each point is in the same cluster as the previous iteration do
    $\forall p \in P$,    cluster($p$) $\leftarrow$ closest $\mu_i$;
    $\forall i \in [1, k]$,    $\mu_i \leftarrow$ mean coordinates of the points in the cluster;
end
```

This procedure is also known to be really sensitive to the initialization of the centroids, and the most common solution is using k-means++ to initialize the centroids [31].

Initializing Lloyd's centroids with the k-means++ algorithm holds significant importance in enhancing the efficiency and effectiveness of the k-means clustering process. Traditional k-means starts with randomly chosen initial centroids, which can lead to suboptimal clustering results and slow convergence, particularly for complex or unevenly distributed data.

In contrast, k-means++ intelligently selects initial centroids based on a probability distribution that prioritizes points that are far apart from each

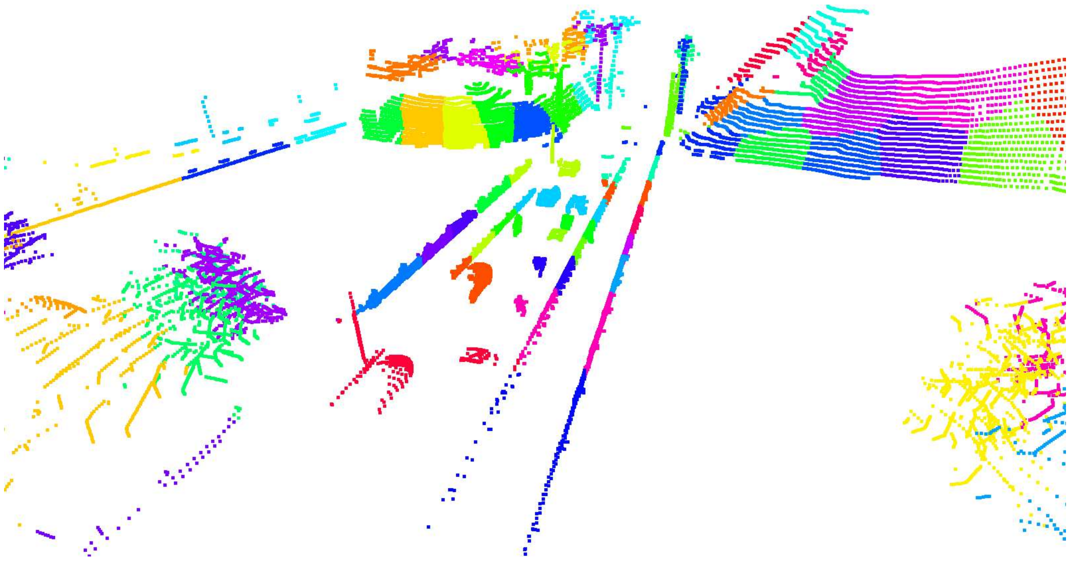


Figure 3.5: Example of clusters on a point cloud

other. This initial placement promotes a more balanced distribution of centroids across the data space, which in turn increases the likelihood of converging to a better final clustering solution.

By strategically initializing centroids using k-means++, the algorithm tends to reach a global or near-global optimum faster, reducing the chances of getting stuck in suboptimal local minima and thus producing more reliable and consistent clustering outcomes.

Furthermore, the k-means++ initialization method contributes to the overall stability and reproducibility of clustering results.

Finally, the identification of clusters related to the same object in different point clouds is necessary for the procedure. The key element to take into account is that mobile elements are moving, but since the acquisition rate of LiDAR is really high, between subsequent acquisitions the shift of mobile objects is really limited.

So, when computing $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$, $\mathbf{P}_t$ clusterization is initialized with k-means++, while $\forall i \in [1, n]$ the clusterization of $\mathbf{P}_{t+i}$ is initialized with the centroids found at $\mathbf{P}_{t+i-1}$. In this way, if clusters are still recognizable, the centroids corresponding to the same object will have the same index in the whole procedure.

Finally, recalling the fact that the clustering procedure on $\mathbf{P}_\tau$ returns an array $\mathcal{C}_\tau$ with k centroids, we can define the dissimilarity measure $\delta_\mathcal{C}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ as the sum of distances between centroids related to the same object,

$$\delta_\mathcal{C}(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = \sum_{i=1}^{k} \|\mathcal{C}_t[i], \mathcal{C}_{t+n}[i]\|_2. \tag{3.12}$$

Intuitively k should be the number of elements present on the street in order to get the best results. Unfortunately, k can not be known a priori on the transmitter, and the computation needed to extrapolate k from the point cloud is too complex to be affordable on this setup.

There are tools from the clustering literature, such as the silhouette coefficient [32], that could solve this problem, but the operations needed the compute the silhouette coefficient introduce a significant overhead in the time needed for the procedure. Meaning, that in this context, hardcode a reasonable value can be the solution.

In Fig. 3.6 and Fig. 3.7, are shown the results of this dissimilarity measure computed on the same set of point clouds used in the other plots of this chapter.

The first important thing to remark on is that this measure is really noisy. Especially when the number of clusters is really low, the noise introduced by the fact that objects are not always correctly identified by the algorithm is really large.

Another important note is that the higher the number of clusters, the higher the time needed to converge to the solution, and this imposes a bound on the number of clusters that are practically computable at the UE.

Finally, here the same problem of voxels can be recognized: $\delta_\mathcal{C}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ depends more on the points of the background than on the points related to the mobile elements, and this explains why the sequence with high traffic have

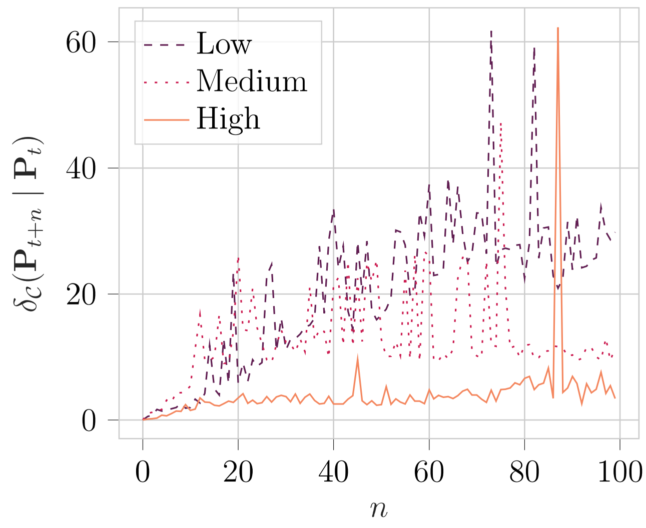


Figure 3.6: Dissimilarity computed using 10 clusters

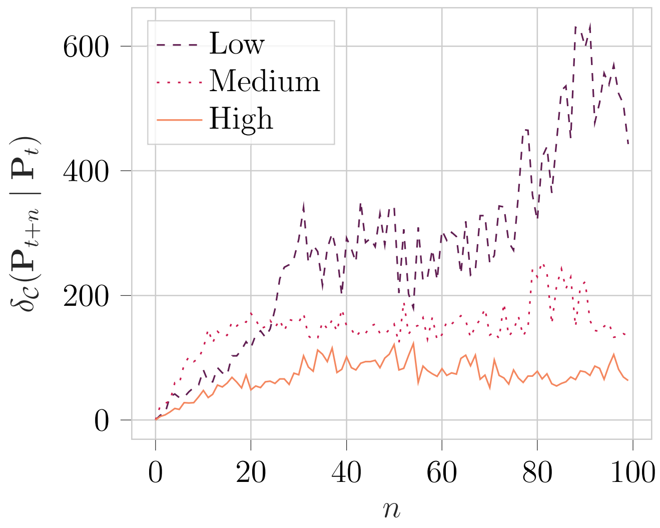


Figure 3.7: Dissimilarity computed using 100 clusters

the lowest dissimilarity.

Furthermore, this procedure violates the assumption of the k-means algorithm, that the cluster must be convex, as there is not any theoretical guarantee on the shape of clusters. In most LiDAR point clouds, objects are not strictly convex.

To overcome this issue, we could use clustering algorithms that do not need the convexity of the cluster. And to this purpose, DBSCAN [33] has been tested, it gives better results (because playing with the definition of distance clusters can be found only on close mobile elements), but is computationally too demanding to be performed at the transmitter.

To sum up, also this definition of dissimilarity does not satisfy our requirements, and, in order to finally find an acceptable approximation of $\delta(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$, a change of perspective is needed to exploit different tools.

3.1.3 Chamfer Distance

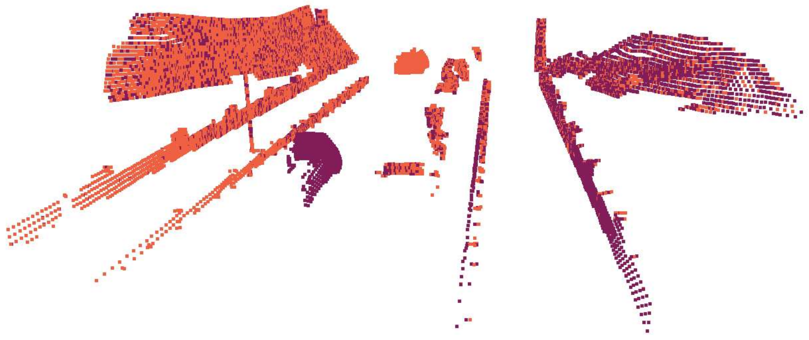


Figure 3.8: Example of Π_τ applied to two point clouds of the same series

To understand the last proposed dissimilarity measure, the starting point is the symmetric point-to-point chamfer distance. This metric has been widely adopted in this context [12], and it is one of the most common dissimilarity measures for point clouds.

The symmetric point-to-point chamfer distance CD_{sym}, measures the dissimilarity between two point clouds. It is a function that takes in input two arbitrary sets of points, without any limitation on the number of points inside each set, referred as $\mathbf{P}$ and $\mathbf{Q}$, and returns a value in $\mathbb{R}^+$. The higher $CD_{sym}(\mathbf{P}, \mathbf{Q})$, the more the two point clouds differ from each other.

To understand how it works, firstly the distance between a point p and a set of points Q must be introduced. The minimum distance between the point and any point belonging to the set will be used as the distance between a point and a set:

$$d(p, \mathbf{Q}) = \min_{q \in \mathbf{Q}} \|p, q\|_2. \tag{3.13}$$

$CD_{sym}(\mathbf{P}, \mathbf{Q})$ is the sum of the distance of all the points in $\mathbf{P}$ from $\mathbf{Q}$ and all the points in $\mathbf{Q}$ from $\mathbf{P}$,

$$CD_{sym}(\mathbf{P}, \mathbf{Q}) = \sum_{p \in \mathbf{P}} d(p, \mathbf{Q}) + \sum_{q \in \mathbf{Q}} d(q, \mathbf{P}). \tag{3.14}$$

The problem in applying directly this dissimilarity measure is that it suffers heavier than other proposed metrics the noise introduced by the background. As previously stated, the number of points related to the background is orders of magnitude bigger than the number of points of mobile objects, so some modifications must be done before using this metric in this context.

First of all, symmetry is not useful for the dissimilarity. In particular, given the point clouds $\mathbf{P}_t$ and $\mathbf{P}_{t+n}$, the sum $\sum_{p \in \mathbf{P}_t} d(p, \mathbf{P}_{t+n})$ represents how far are the old points to the new point cloud. This is related to the old information, but as stated before there is no need to take into account objects that are no longer present.

On the other side, to overcome the problem related to the points in the background a filtering action is needed. The most promising solution is pruning every point that is further than 20 m from the position of the sensor and below the level of the street. In this way, a lot of outliers are removed and the measure is less noisy.

To sum up, the new pruning function is defined as

$$\Pi_\tau(\mathbf{P}) = \{p \text{ if } p_z > -p_l + \varepsilon \quad \forall p \in \mathbf{P} \text{ and } \|p, p_s\|_2 < 20\}, \tag{3.15}$$

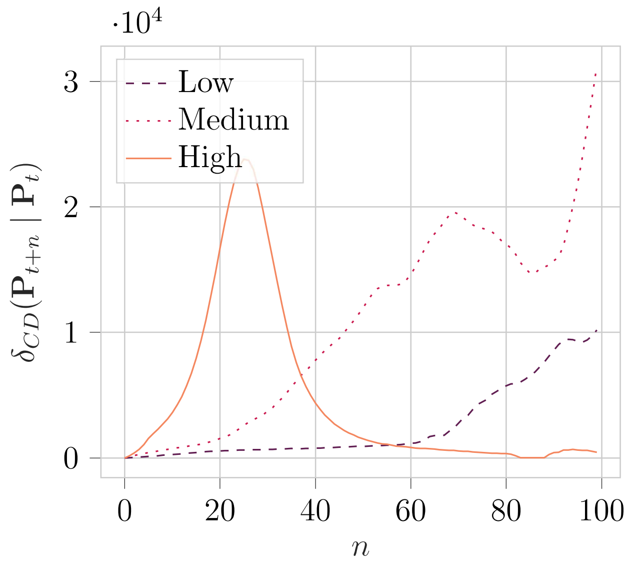


Figure 3.9: Dissimilarity computed using $\delta_{CD}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$

with p_l relative position of the LiDAR with respect to the car and p_s absolute position of the sensor at time τ.

A new dissimilarity measure can then be defined:

$$\delta_{CD}(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = \sum_{p \in \Pi_{t+n}(\mathbf{P}_{t+n})} d(p, \Pi_{t+n}(\mathbf{P}_t)). \tag{3.16}$$

$\delta_{CD}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ was tested on the same point clouds of this chapter. This metric is indeed able to capture the difference between different levels of traffic as reported in Fig. 3.9.

The only problem is that this measure has a periodic behavior. In a standard urban environment, a set of vehicles is moving forming a lane. Fixing the initial time t, and the positions of all vehicles in the lane at this instant, δ_{CD} increases until vehicles are in the furthest positions from the ones recorded at time t. But then, since cars are moving in a lane, after reaching the maximum distance, they get closer to the positions registered at time t, and therefore the δ_{CD} decreases.

To overcome this problem and make the dissimilarity non-decreasing a cumulative sum version of $\delta_{CD}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ can be defined,

$$\hat{\delta}(\mathbf{P}_{t+n} \mid \mathbf{P}_t) = \sum_{i=1}^{n} \delta_{CD}(\mathbf{P}_{t+n} \mid \mathbf{P}_t). \tag{3.17}$$

In Fig. 3.10 is plotted the evolution of this metric. The biggest problem is that after some time it diverges from the expected behavior.

In this context, expecting to send point clouds periodically and therefore bounding the maximum value that n can take, $\hat{\delta}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$ is a sufficiently good approximation of the dissimilarity.

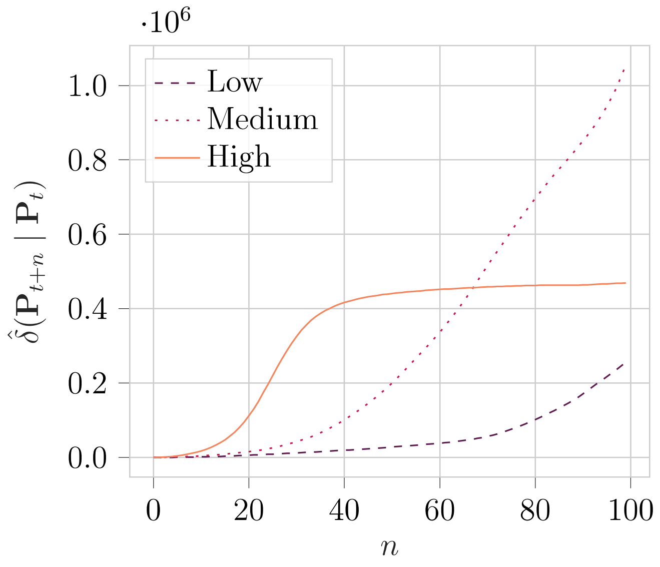


Figure 3.10: Dissimilarity computed using cumulative $\hat{\delta}(\mathbf{P}_{t+n} \mid \mathbf{P}_t)$

Chapter 4

PQoS Framework

In the previous chapter, the emphasis was on determining which point clouds were worth transmitting for the purpose of TD, while others were deemed redundant. Now, the attention shifts towards compression. The underlying query for this chapter is: “To what extent point clouds can be compressed while retaining their essential information?”

To address this query, the initial step is to introduce the complete framework that underpins this work.

In the TD scenario vehicles are equipped with some onboard sensors, each vehicle is capable of performing simple operations and it aims to transmit data to an edge server.

The edge server is then responsible for deciding how the car should move.

To programmatically decide how the car should move the edge server must be able to correctly identify objects in the scene. And from a computational point of view, this operation is demanding in terms of time and resources. For this reason, this can not be done at the UE. Assuming to have more computational power on the edge server, this operation is easily done if the task is offloaded.

On the other side, offloading this operation is time-consuming due to the transmission. Raw point clouds are big unordered data structures that are not easy to effectively compress in a lossless fashion.

There are plenty of different methods to compress point clouds, but only a few of them are suitable for this application. G-PCC [34], for example, even if can reach really high levels of compression is not suitable for this application, because this compression is heavily time-consuming.

In other studies [35], Draco, a 3D compression algorithm, has been identified as one of the most promising solutions to tackle this problem.

This chapter will deeply analyze the performances achievable with different levels of compression, trying to find the best trade-off between compression and accuracy.

4.1 PointPillars and mean Average Precision

First, is important to understand the main metric for the framework which is the mAP. The mAP is generally used to compute the quality of a prediction. However, since the quality of the prediction in this framework is dependent on all the operations involved, the mAP is the most suitable metric to measure the QoE experienced with this framework.

mAP is a widely used evaluation metric in the field of computer vision, specifically for assessing the performance of object detection algorithms [36]. It quantifies the accuracy of an object detection system by considering both precision and recall across multiple levels of confidence thresholds.

To understand the mAP, the knowledge of precision and recall is needed. Precision is the ratio of true positive detections (objects that are correctly identified) to the total number of positive detections (true positives and false positives). It measures the accuracy of positive predictions. Recall, on the other hand, is the ratio of true positive detections to the total number of ground truth objects (true positives and false negatives). It measures the ability to correctly identify all relevant objects.

The precision-recall curve is created by varying the confidence threshold for positive predictions and computing precision and recall values at each threshold. Average Precision (AP) is then calculated by computing the area under this precision-recall curve. It summarizes the algorithm's ability to balance precision and recall across different levels of confidence thresholds.

In the case of study, there are multiple object classes that need to be detected. mAP extends the concept of AP to account for multiple classes. For each class, AP is computed independently, and then the mean of these AP values is taken to compute the overall mAP. This accounts for variations in detection performance across different classes.

Object detection frameworks often use mAP as a benchmark to compare the effectiveness of different algorithms. Higher mAP scores indicate better performance in accurately localizing and classifying objects.

In this scenario, the process of classifying detections as true positives, false positives, or false negatives involves assessing the spatial overlap between predicted bounding boxes and ground truth bounding boxes. This is typically achieved using a measure called Intersection over Union (IoU).

IoU is a metric used to quantify the degree of overlap between two bounding boxes. It is calculated as the ratio of the volume of the intersection between the two boxes to the volume of their union. In object detection, if the IoU between a predicted bounding box and a ground truth bounding box exceeds a certain threshold (in this case 0.5), the prediction is considered a true positive. If the IoU is below the threshold, the prediction is treated as a false positive. Ground truth bounding boxes that do not have corresponding predictions are labeled as false negatives.

Since the precision-recall curve may not be densely sampled at all possible confidence thresholds, the mAP is often calculated by interpolating the precision values at a finite set of recall levels. This interpolation process yields a smoother curve and provides a more stable measure of performance. This approach is known as interpolation of precision at different recall levels, in this work 11 recall levels are used, evenly spaced in $[0, 1]$.

The mAP is particularly suitable for this analysis because it is a comprehensive metric, that represents the accuracy obtained with the entire procedure. It is important to notice that the mAP is only related to object detection, but the outcome of the object detector is influenced by each step in the procedure, so the mAP can be used as an end-to-end metric for the framework.

At the receiver, in order to compute the location of objects inside the transmitted point cloud, the object-detection is implemented with PointPillars [37].

PointPillars is designed to solve the challenges of detecting objects in complex 3D environments, such as those encountered in TD.

In PointPillars point clouds are preprocessed into a new representation before undergoing the process of detection. By exploiting a structured grid representation PointPillars can achieve high precision and fast detection.

During this preprocessing, a regular grid is applied to the x-y plane of the point cloud. The grid is made of parallel planes with a fixed distance. This distance takes the name of pillar size ρ.

This grid defines a set of parallelepipeds known as "pillars." Information coming from all points inside each pillar is aggregated, forming a bidimensional representation of the point cloud.

In essence, the process of pillar creation in PointPillars involves transforming raw point cloud data into a structured grid format, where each grid cell embodies information about the objects in the scene. This is crucial when combined with the compression, and an analysis of the interaction between pillars and the compression is proposed in Sec. 4.2.

Once this representation is obtained, the detection is done using a series of convolutional layers, to first extract features from the grid representation and then to locate objects. To speed up the computation PointPillars takes advantage of sparse convolutional layers, since in most pillars there are not points and therefore there is no information in the pillar.

Finally, the neural network architecture of PointPillars is tailored for multitasking. It predicts the presence of objects within each grid cell while concurrently refining bounding box parameters if an object is detected. This dual functionality eases the object detection pipeline and contributes to the network's performance.

PointPillars has been trained and tested on the KITTI dataset [38]. Since SELMA has features that are really different from the ones of KITTI, retraining the algorithm on the new dataset was necessary.

Furthermore, since samples are assumed independent and identical distributed (iid) during the training, PointPillar was not trained with the version of SELMA presented in this thesis, but an older version specifically designed for this purpose [19]. This version contains samples of various acquisitions, but samples are not time correlated as in the case of the new SELMA.

The old version of SELMA was split into three sets, the train set, used to provide samples to the algorithm for learning, the validation set, used to monitor the learning and interrupt the learning before overfitting, and the test set to assess the goodness of the procedure.

Pillar size ρ (m)	**mAP**
0.16	0.669
0.50	0.704
1.28	0.393

Table 4.1: mAP achievable changing Pillar Size

As stated before the original pillar size was changed to use PointPillars in this context. In particular, the algorithm has been trained several times and the most promising results in terms of mAP are shown in Tab. 4.1.

4.2 Draco Compression

Draco is a high-performance and tunable open-source library developed by Google for compressing and decompressing 3D geometric data, encompassing both meshes and point clouds. It was engineered to optimize both compression efficiency and processing speed, offering a solution for improving the transmission of 3D objects.

Draco functions as an adaptive and versatile compression framework, allowing balancing between compression ratios and decoding performance according to their specific use cases. This tunability grants users the flexibility to adapt Draco's compression settings to suit their application's requirements.

When using Draco to compress a point cloud you can tune mainly two parameters: the number of quantization bits q and the compression level c.

As the name suggests, the number of quantization bits q is the number of bits that Draco will use to encode each coordinate of each point in the point cloud. Since LiDARs in SELMA have a range of 100 m, the minimum distance we can encode with q bits of quantization is:

$$\Delta = 200/2^q. \tag{4.1}$$

And to give an example, with $q = 10$ the resolution is approximately 20 cm.

This parameter is the one that most affect the size of the file and also the quality of the compression, since for Draco the higher part of the compression error is coming from the quantization.

On the other side, the c parameter, which can take values in $[1, 10]$, controls the operations performed on the point clouds and the byte streams when encoding the point clouds. The higher c, the higher the level of compression, and the higher the time needed to encode or decode the point cloud.

In order to estimate this behavior some tests were performed on the point clouds generated for SELMA. In Fig. 4.1 you can compare the time it takes to compress a point cloud depending on the q and c used.

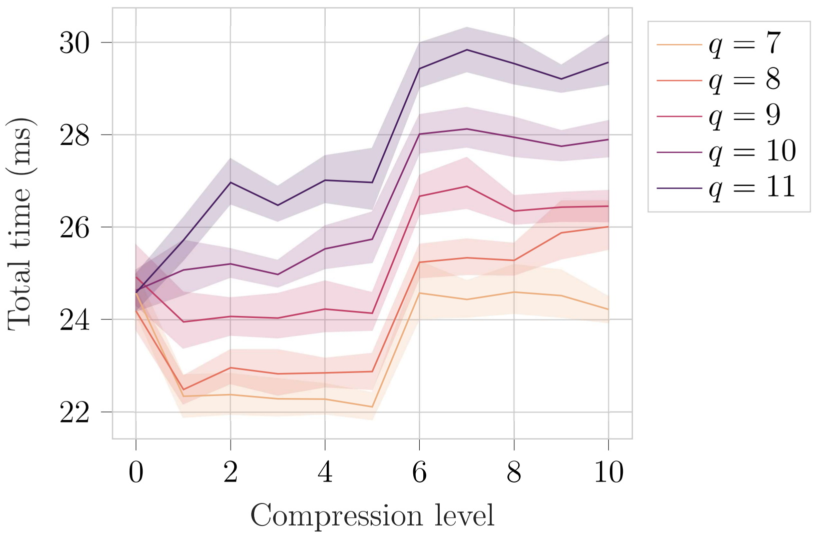


Figure 4.1: Total time needed to encode and decode a point cloud using Draco

When feeding PointPillar with a point cloud that was compressed with Draco, the minimum distance between points Δ takes a crucial importance. In particular, if the distance Δ is bigger than the size of a pillar ρ, there will be empty pillars even in regions where the point cloud is dense.

Due to the convolutional nature of PointPillar, then, the presence of empty pillars inside the representation of the point cloud will heavily alter the capability of convolutional layers to correctly identify features in this representation.

This leads to the identification of a threshold $\hat{q}$ for the number of quantization bits. The threshold can be computed as

$$\hat{q} = \left\lfloor \log_2 \frac{200}{\rho} \right\rfloor, \tag{4.2}$$

that is the highest q for which $\Delta > \rho$

If the quantization is done with $q > \hat{q}$ the error introduced by the compression is not enough to alter the performances of PointPillars, therefore the mAP that can be achieved when compressing and detecting point cloud is constant

$\forall q > \hat{q}$.

But on the other side, when $q \leq \hat{q}$ the quality of the prediction degrades really fast, reaching a mAP close to zero with small changes in q.

In Fig. 4.2 is plotted the mAP achievable with different choices of q, c and ρ. Since the version of PointPillars with $\rho = 50$ cm outperforms every other version for each level of compression, from now on only the version with $\rho = 50$ cm will be used. Leading to the fact that the threshold for having $\Delta > \rho$ is reached when $\hat{q} = 8$.

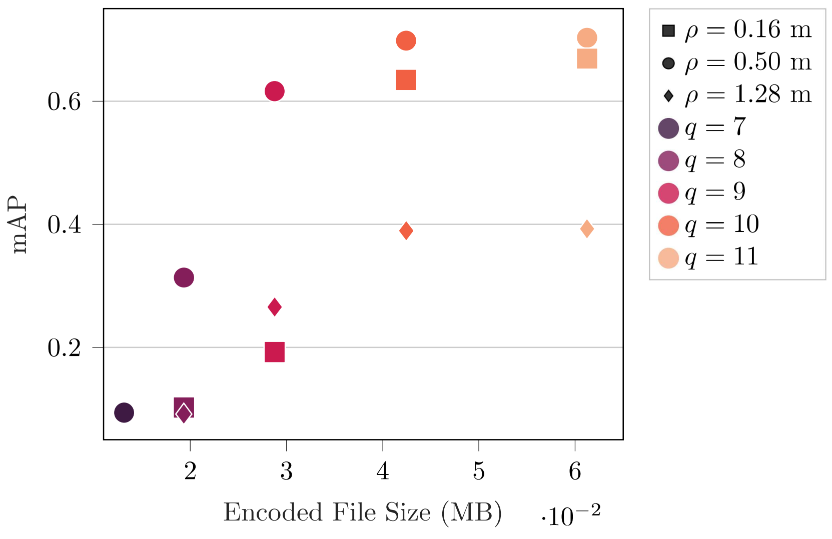


Figure 4.2: mAP of PointPillar using point clouds compressed with Draco

4.3 Double Deep Q-Learning Network

In this framework, a RL agent controls in real-time the compression level used by Draco. Given the tradeoff between the mAP and the amount of data transmitted over the network, the RL agent has to maximize the achieved mAP while respecting the KPIs imposed by the TD.

A RL model characterizes the environment (the network), as a Markov Decision Process (MDP), where time is discretized into steps $t = 0, 1, 2, \ldots$.

At each step t, the agent observes the environment state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$, with state and action spaces $\mathcal{S}$ and $\mathcal{A}$, respectively. The environment transitions to a new state $s_{t+1} \in \mathcal{S}$ based on s_t and a_t, and the agent receives a reward $r_t \in \mathbb{R}$.

The objective in RL is to find the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maximizes the cumulative sum of rewards

$$\sum_{\tau=t}^{+\infty} \gamma^{\tau-t} r_\tau \tag{4.3}$$

over time, where $\gamma \in [0, 1]$ is the discount factor.

This is in some cases done by approximating a function called $Q_\pi(s, a)$, this function expresses the expected cumulative reward that the agent will get by taking the action a in the state s and then following the policy π.

The D-DQN algorithm [39] was employed to learn the optimal policy. This algorithm is characterized by having two similar Deep Neural Network (DNN), called primary and target networks.

The primary network, the first DNN, is trained to approximate the Q-function, using as policy the agent's policy π. Given a state s_t it produces Q-values $Q_\pi(a_t, s_t)$ for each action $a_t \in \mathcal{A}$ based on the current state s_t. And upon those Q-values, the agent decides which action $a_t \in \mathcal{A}$ has to be performed at the time step t.

The D-DQN algorithm introduces a target network, an asynchronous DNN variant of the primary network. The target network's parameters are updated every υ learning steps. The second network estimates Q-values using an old version of the primary network to prevent positive feedback that would lead to an overestimation of the Q-values, known as "overestimation bias", ensuring more robust learning.

Statistical learning has been used to train the primary network to approximate the Q-function. In particular, the Stochastic Gradient Descent (SGD) technique has been implemented. SGD requires iid data, but, since samples from the network are temporally correlated, a memory replay approach must be adopted to break the correlation between samples.

In the memory replay approach, the learning starts only when μ transitions (s_t, a_t, r_t, s_{t+1}) have been recorded. The agent randomly selects β transitions from the memory to break the correlation between subsequent acquisitions, and those transitions are used to update the primary network. Older transitions are removed when memory is full to prioritize recent observations.

One of the significant problems with RL is that once the agent finds a suitable policy it tends to repeat this policy avoiding exploring other alternatives. To encourage exploration, an ϵ-greedy policy was implemented. At each step t, the agent selects a random action $a_t \in \mathcal{A}$ with probability ϵ (exploration), and with probability $1-\epsilon$, it chooses the best action (exploitation); ϵ linearly decreases during training.

The RL model comprises the state and reward function. The state s includes measurements from the network. The reward r depends on QoS and QoE, with QoS enforcing end-to-end communication delay $\delta_{\text{APP},t}$ lower than δ_{M}. QoE considers accurate data transmission for driving tasks and is proportional to the mAP achieved in the time step t.

Being $\delta_{\text{APP},t}$ the maximum APP delay experienced during the time slot t, $\overline{\delta_{\text{APP}}}$ the requirement in terms of application delay to perform TD, $\text{PRR}_{\text{APP},t}$ the packet reception rate on the same time slot and $\overline{\text{PRR}_{\text{APP}}}$ the requirement in terms of packet receprion rate to perform TD, and $\text{mAP}(a_t)$ the average mAP obtained by choosing the action a_t, the reward r_t at step t is defined as follows:

$$r_t = \begin{cases} -e^{\frac{\delta_{\text{APP},t} - \overline{\delta_{\text{APP}}}}{\overline{\delta_{\text{APP}}}}} + 1 & \delta_{\text{APP},t} > \overline{\delta_{\text{APP}}} \\ -e^{\overline{\text{PRR}_{\text{APP}}} - \text{PRR}_{\text{APP},t}} + 1 & \text{PRR}_{\text{APP},t} < \overline{\text{PRR}_{\text{APP}}} \\ \text{mAP}(a_t) - \varepsilon \delta_{\text{APP},t} & \text{otherwise} \end{cases} \tag{4.4}$$

with

$$\varepsilon < min_{a,a' \in \mathcal{A}} \mid \mathrm{mAP}(a) - \mathrm{mAP}(a') \mid . \tag{4.5}$$

The realization of the reward as a function of the delay is reported in Fig. 4.3.

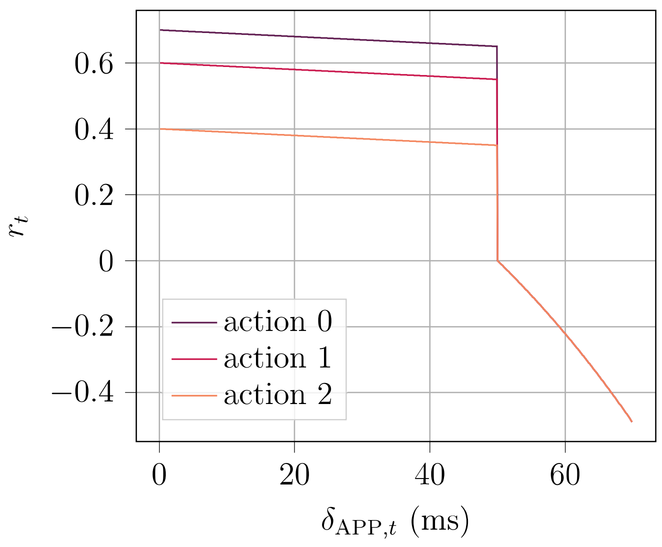


Figure 4.3: Reward function changing the delay

In this case the $\overline{\delta_{\mathrm{APP}}}$ is set to 50 ms, it is clear that if KPIs are addressed during the time step t the reward r_t is positive and it is mainly based on the action chosen a_t. The linear behavior is only imposed by the need to avoid zero gradients.

On the other side if KPIs are not addressed the reward does not depend on the action, is negative, and goes down exponentially when the delay increases or the packet reception rate decreases.

4.4 Simulation using ns3

Once all the components of the framework have been presented, the complete detailed structure of the framework can be introduced.

At the UE, point clouds of the surroundings are acquired by a LiDAR sensor. This point cloud is then preprocessed and sent to a Next Generation Node Base (gNB) where an edge server computes the driving action that the vehicle must actuate. This information is then sent to the UE in order to actuate it.

The UE, during the preprocessing of the point cloud, compresses the point cloud using Draco. The compression level c and the number of quantization bits q are chosen by a RL agent that implements a D-DQN algorithm.

The edge server, on the other hand, uses PointPillars to detect objects in the point cloud compressed by the UE, the mAP obtained by this operation is then used to train the D-DQN algorithm at the UE.

The D-DQN can choose between three different actions reported in Tab. 4.2. The state of the D-DQN contains 18 entries, with information on the whole protocol stack. In particular, from the Physical (PHY) layer, the metrics chosen are the Modulation and Coding Scheme (MCS), transmitted Orthogonal Frequency Division Multiplexing (OFDM) symbols, and average Signal to Interference plus Noise Ratio (SINR). From the Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), and Application (APP) layers the delay and the packet reception rate are used.

q	c	**mAP**
10	10	0.683
9	10	0.575
8	10	0.257

Table 4.2: Action space $\mathcal{A}$ of the D-DQN

In order to get realistic data and results, the communication is simulated using ns3, an open-source discrete-event network simulator for Internet systems.

This extends the RAN-UEAI framework presented in [40] and [12], which introduces the framework and decentralizes PQoS functionalities respectively.

The goal is to introduce a more realistic setting, dropping some assumptions that were made in previous works.

The application module presented in [40] has been adapted to this new setup. The application aimed to simulate data exchange and generate sensor data. This generation process is influenced by three key factors:

1. The size of the initial sensor data, measured in bytes.

2. The regularity with which data is generated and exchanged.

3. The chosen degree of compression applied to the sensor data.

In this study, sensor data extracted from the SELMA dataset and the Draco compression algorithm are used.

With the same rationale of `KittiTraceBurstGenerator`, the `TraceFileBurstGenerator` has been extended into the `SelmaTraceBurstGenerator`. This class allows the user to reproduce real-world traffic traces following data samples from SELMA.

The `SelmaTraceBurstGenerator` is installed on each UE to simulate the traffic flow generated by LiDARs acquisitions.

On the other side, the UE-AI, responsible for adapting the transmission mode to the network state, is implemented through the `UeAI` and `MmWaveUeNetDevice` classes that facilitate distributed PQoS. Those classes are integrated into the `mmwave` module in ns-3.

The main functionalities of those classes are:

- Initialization: The `InstallUserAI` method installs UE-AI, initializes measurement collection, and schedules updates.

- Measurement Collection: The `RxPacketTraceUe` method captures PHY metrics, including MCS, transmitted OFDM symbols, and average SINR. The `SendStatusUpdate` method records metrics from the RLC, PDCP, and APP layers.

- Network Control: The `SendStatusUpdate` method reports measurements to the RL agent, determining optimal compression levels with the D-DQN.

- Application Control: The `NotifyActionIdeal` method conveys the RL agent's decision to the application for configuration.

Finally, to keep a fair environment and reduce the time needed to reach convergence, the D-DQN algorithms are trained in a federated fashion [41]. This means that, with periodic updates, the weights of the DNN are shared with the gNB. The gNB that performs operations to combine them and redistributes the model to sum all the experience gathered from different devices and to ensure the fact that all UEs are following the same policy π.

Chapter 5

Performance Evaluation

The PQoS framework presented in Chapter 4 has been trained and tested in different conditions, in particular varying the number of vehicles in the system. The parameters used in the simulations are reported in Tab. 5.1.

Specifically, the D-DQNs have been trained on a set of traces containing the 90% of the total traces, and tested on the remaining 10%. The training phase consists of 1000 episodes, or ns-3 simulations, of 80 s each. Each episode is then divided into 800 steps of 100 ms each. During the testing phase, on the other side, 100 episodes have been performed without training the neural network.

5.1 Results with One Vehicle

In this section, we present some simulation results in which only one vehicle is deployed. In Fig. 5.1 we report the action probability, i.e., the normalized frequency with which each action is taken in each episode during the training phase.

From this plot, it is clear that the D-DQN algorithm is exploring different actions. Eventually, compression with $q = 10$ quantization bits is the most preferred action; still, $q = 9$ has a non-negligible probability of being used,

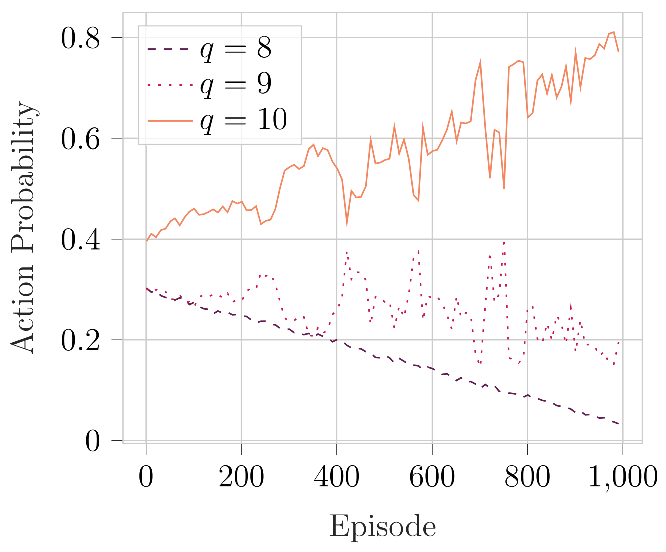


Figure 5.1: Action probability during the training phase of the D-DQN.

while $q = 8$ is never hardly selected. This indicates that the D-DQN algorithm is trained to choose when to change between $q = 9$ and $q = 10$ to meet the KPIs, and learns not to choose $q = 8$, meaning that in most cases the algorithm is able to address the KPIs of TD using compression levels that preserve most of the information in the point clouds. Indeed, the mAP increases as increasing

Parameter	Description	Value
f_c	Carrier frequency	3.5 GHz
B	Total bandwidth	50 MHz
P_{TX}	Transmission power	23 dBm
n	Number of vehicles	$\{1, 3, 5, 8\}$
$\overline{\delta_{\text{APP}}}$	Max. tolerated delay	50 ms
$\text{mAP}_{q=8}$	mAP for the action $q = 8$	0.257
$\text{mAP}_{q=9}$	mAP for the action $q = 9$	0.575
$\text{mAP}_{q=10}$	mAP for the action $q = 10$	0.683
r	LiDAR perception rate	10/s
γ	Discount factor	0.95
υ	Update interval	0.1 s
μ	Memory replay size	$8 \cdot 10^4$ B
β	Batch size	32 B
ζ	Learning rate	10^{-5}

Table 5.1: Scenario parameters.

Policy	KPIs probability	Average δ_{APP}	Average mAP	Average r_t
D-DQN	86.2%	26.5 ms	0.680	$-4.8 \cdot 10^6$
$q = 8$	90.4%	23.2 ms	0.257	$-6.1 \cdot 10^8$
$q = 9$	87.7%	25.4 ms	0.575	$-4.5 \cdot 10^9$
$q = 10$	84.4%	28.0 ms	0.683	$-3.5 \cdot 10^{10}$

Table 5.2: Comparison between D-DQN and some constant policies as a function of several metrics.

the number of quantization bits.

Notably, results in Fig. 5.1 are statistically sound, and empirically demonstrate the technical accuracy of our D-DQN framework to support PQoS in the TD scenario.

Moreover, in Tab. 5.2 we report, respectively, the percentage of time slots in which the KPIs are satisfied, the average e2e delay δ_{APP} (which is a measure of the QoS) experienced at the APP level, the average mAP (which is a measure of the QoE, expressed as the quality of the object detection operation), and the average reward. The D-DQN algorithm is compared to some constant benchmark policies where the compression level of the point clouds is fixed a priori for the whole duration of the simulation. From Tab. 5.2, the trade-off between QoS and QoE is clear, and the D-DQN algorithm is able to finely balance between the two achieving a very good mAP while respecting the KPIs for 86.2% of the time. The only policy with a significantly higher KPIs probability is $q = 8$, at the expense of a very bad mAP of only 0.257. Overall, the average reward of D-DQN is superior to any of the other policies for PQoS.

Another interesting fact is that the D-DQN algorithm, also when KPIs are not met, achieves an average delay of 77.5 ms, which is lower than any of the other constant benchmarks.

While the values in Tab. 5.2 are on average, to better capture the variability of network metrics among the different policies, in Figs. 5.3 and 5.2 we report the boxplots of the delay and of the rewards statistics achieved during the

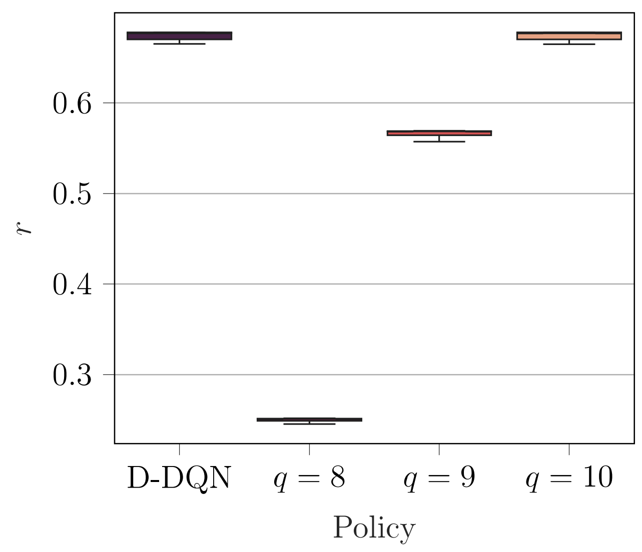


Figure 5.2: Distribution of the reward for D-DQN vs. the constant policies, during the testing phase.

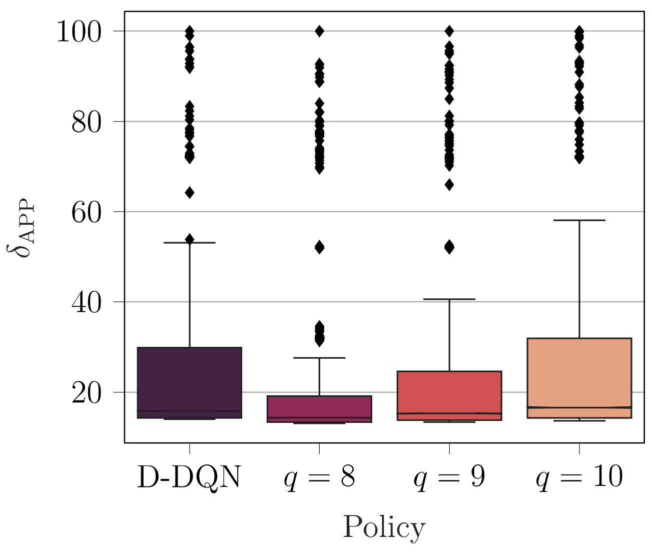


Figure 5.3: Distribution of the delay for D-DQN vs. the constant policies, during the testing phase.

testing phase. From Fig. 5.2 it is clear that the reward achieved by D-DQN is statistically really similar to the one for the constant policy $q = 10$, and is higher than any of the other benchmarks. In terms of delay, in Fig. 5.3 we see that D-DQN has a similar performance to the constant policies $q = 8$ and $q = 9$ (which in turn suffer in terms of mAP), while outperforms $q = 10$,

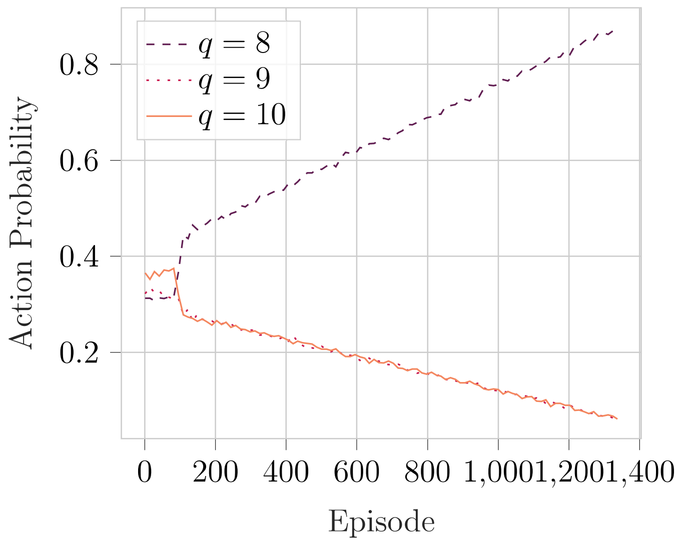


Figure 5.4: Action probability during the training phase of the D-DQN.

especially in the higher percentiles. These results enforce the hypothesis that the D-DQN algorithm is effectively learning how to change the compression based on the network status and maximize both QoS and QoE simultaneously.

5.2 Analysis with Multiple Vehicles

In this subsection, we consider a scenario with multiple vehicles, specifically 3, 5, and 8 vehicles, even though results will be proven to be similar.

First, in Fig. 5.4 we analyze the evolution of the action probabilities during the training phase considering 8 vehicles. Due to the federated learning approach in D-DQN, and due to the fact that vehicles experience similar network conditions, results would be the same also considering 3 and 5 vehicles. Notably, D-DQN tends to only choose action $q = 8$, i.e., the action that compresses the point cloud the most, even though this approach would irreparably damage the mAP. This is due to the fact that, as the number of vehicles increases, the network becomes more congested, and users are required to reduce the size of the data to send as much as possible to alleviate the burden on the channel. In these conditions, the D-DQN is not even trying to balance between

N. of Vehicles	KPIs probability	Average δ_{APP}	Average mAP
1	86.2%	26.5 ms	0.680
3	73.9%	36.8 ms	0.257
5	51.3%	55.4 ms	0.257
8	27.7%	73.7 ms	0.257

Table 5.3: Performance of D-DQN under several metrics, as a function of the number of vehicles.

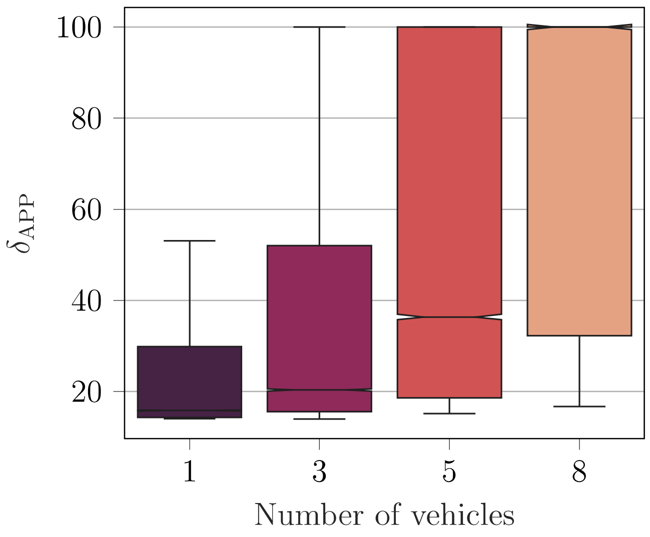


Figure 5.5: Distribution of the delay for D-DQN varying the number of vehicles, during the testing phase.

different compression levels, and behaves as the constant policy $q = 8$.

Results as a function of the number of vehicles for D-DQN are reported in Tab. 5.3. Unfortunately, we see that the network is not able to support PQoS as the number of vehicles increases, regardless of the D-DQN implementation and even considering the most aggressive compression configuration. In particular, the probability of satisfying the KPIs drops to only 27% with 8 vehicles, and the average delay is as large as 73 ms, which is far beyond the requirements of TD applications. We have the same results considering the APP delay experienced during the testing phase, as reported in Fig. 5.5.

This is not due to how D-DQN has been structured, but rather to the limited network resources available for transmission in the current simulation scenario. In order to overcome this issue, there are several options.

First, while D-DQN is currently designed to only optimize the compression level, that is the size of data packets, it could be changed to adjust the transmission rate too, for example, based on the results in Chapter 3. In particular, D-DQN can be trained to transmit only relevant or critical point clouds with respect to what is already available at the receiver(s), thus reducing the transmission rate below that of the sensors. This approach would drastically reduce the amount of data to be sent through the network, thus improving the latency.

Second, D-DQN can be designed to operate at the RAN level too, e.g., using a higher transmission power to increase the SINR and send data faster when needed. In addition, a higher numerology would increase the number and size of resource blocks available for transmission, as well as the total bandwidth, which would improve the performance of the entire system.

Third, a better compression algorithm and/or object detector might be able to improve the mAP even with very aggressive compression configurations, which would allow for further reduction of the number of bits in the point clouds with minor degradation of the quality of the point cloud itself.

Chapter 6

Conclusions

In this thesis, we considered a TD scenario in which vehicles are remotely controlled by a teleoperator that relies on data acquired by the vehicles' sensors to make decisions on the vehicles' movements.

One of the main research concerns for TD is related to the variability of the network. Indeed, given the strict KPIs that must be respected to ensure safety in this scenario, it is important to predict and control changes in the QoS, and react accordingly.

In this context, PQoS was identified as a suitable technique to ensure that KPIs are satisfied. Specifically, PQoS is able to predict the status and evolution of the network, and use these predictions to take countermeasures at the RAN level in case KPIs are not satisfied.

In this thesis, two approaches have been presented to tackle this problem. The first approach involves goal-oriented communication, where data transmission probability is proportional to data value. The second approach explores the trade-off between QoS and QoE by adjusting point cloud compression modes.

Data availability for training is essential for both approaches. This work utilized and expanded the SELMA dataset for autonomous driving, which contains synthetic sensor data from vehicles in urban settings. This dataset is built upon the CARLA simulator and incorporates techniques to control pedestrian, vehicle, and map behavior.

Using SELMA, we first focused on point cloud correlation. Techniques for approximating point cloud correlation onboard the vehicles have been proposed, involving voxel representation correlation, clusterization, and a customized Chamfer Distance metric. The latter showed promising performance in accurately measuring point cloud correlation.

Then, a comprehensive end-to-end framework for PQoS has been introduced. Specifically, we designed an RL agent, trained using federated learning and a reward based on application KPIs, to determine the optimal compression levels for point clouds. The compressed data is sent to an edge server for object detection using PointPillars, with detection quality measured in terms of mAP, which is used to train the RL agent.

Performance evaluation via ns-3 simulations demonstrates the effectiveness of the RL agent. However, we proved that performance degradation occurs as the number of vehicles increases due to potential transmission resource limitations.

These results motivate further research efforts in this domain. For example, we will integrate the analysis in Chapter 3 in the framework proposed in Chapter 4. This integration will reduce the burden on the network by jointly optimizing the data rate (based on the level of correlation) and size (based on the level of compression) of point clouds. Also, we will explore different numerologies for the transmission of the point clouds, to increase the capacity of the network.

Bibliography

[1] S. Neumeier, N. Gay, C. Dannheim, and C. Facchi, “On the Way to Autonomous Vehicles Teleoperated Driving,” in *AmE - Automotive meets Electronics; 9th GMM-Symposium*, 2018.

[2] C. Kettwich, A. Schrank, H. Avsar, and M. Oehl, “What If the Automation Fails? – A Classification of Scenarios in Teleoperated Driving,” in *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI ’21 Adjunct, (New York, NY, USA), p. 92–96, Association for Computing Machinery, 2021.

[3] J.-M. Georg, J. Feiler, F. Diermeyer, and M. Lienkamp, “Teleoperated Driving, a Key Technology for Automated Driving? Comparison of Actual Test Drives with a Head Mounted Display and Conventional Monitors,” in *21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3403–3408, 2018.

[4] M. Hofbauer, C. B. Kuhn, G. Petrovic, and E. Steinbach, “TELECARLA: An Open Source Extension of the CARLA Simulator for Teleoperated Driving Research Using Off-the-Shelf Components,” in *IEEE Intelligent Vehicles Symposium (IV)*, pp. 335–340, 2020.

[5] S. Neumeier, E. A. Walelgne, V. Bajpai, J. Ott, and C. Facchi, “Measuring the Feasibility of Teleoperated Driving in Mobile Networks,” in *Network Traffic Measurement and Analysis Conference (TMA)*, pp. 113–120, 2019.

[6] M. Boban, M. Giordani, and M. Zorzi, "Predictive Quality of Service: The Next Frontier for Fully Autonomous Systems," *IEEE Network*, vol. 35, no. 6, pp. 104–110, 2021.

[7] D. C. Moreira, I. M. Guerreiro, W. Sun, C. C. Cavalcante, and D. A. Sousa, "QoS Predictability in V2X Communication with Machine Learning," in *IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020.

[8] N. Duffield, J. Lewis, N. O'Connell, R. Russell, and F. Toomey, "Predicting quality of service for traffic with long-range fluctuations," in *Proceedings IEEE International Conference on Communications ICC*, vol. 1, pp. 473–477 vol.1, 1995.

[9] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," *Mobile networks and applications*, vol. 25, pp. 391–401, 2020.

[10] F. Mason, M. Drago, T. Zugno, M. Giordani, M. Boban, and M. Zorzi, "A Reinforcement Learning Framework for PQoS in a Teleoperated Driving Scenario," in *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 114–119, 2022.

[11] Z. Wu and D. Yan, "Deep reinforcement learning-based computation offloading for 5G vehicle-aware multi-access edge computing network," *China Communications*, vol. 18, no. 11, pp. 26–41, 2021.

[12] F. Bragato, T. Lotta, G. Ventura, M. Drago, F. Mason, M. Giordani, and M. Zorzi, "Towards Decentralized Predictive Quality of Service in Next-Generation Vehicular Networks," *IEEE Information Theory and Applications Workshop (ITA)*, 2023.

[13] O. Goldreich, B. Juba, and M. Sudan, "A Theory of Goal-Oriented Communication," *J. ACM*, vol. 59, may 2012.

[14] E. Calvanese Strinati and S. Barbarossa, "6G networks: Beyond Shannon towards semantic and goal-oriented communications," *Computer Networks*, vol. 190, 2021.

[15] N. Pappas and M. Kountouris, "Goal-Oriented Communication For Real-Time Tracking In Autonomous Systems," in *IEEE International Conference on Autonomous Systems (ICAS)*, 2021.

[16] F. Pezone, S. Barbarossa, and P. Di Lorenzo, "Goal-Oriented Communication for Edge Learning Based On the Information Bottleneck," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8832–8836, 2022.

[17] F. Binucci, P. Banelli, P. Di Lorenzo, and S. Barbarossa, "Dynamic Resource Allocation for Multi-User Goal-oriented Communications at the Wireless Edge," in *30th European Signal Processing Conference (EUSIPCO)*, pp. 697–701, 2022.

[18] M. Giordani, T. Higuchi, A. Zanella, O. Altintas, and M. Zorzi, "A Framework to Assess Value of Information in Future Vehicular Networks," in *Proceedings of the 1st ACM MobiHoc Workshop on Technologies, MOdels, and Protocols for Cooperative Connected Cars*, TOP-Cars '19, (New York, NY, USA), p. 31–36, Association for Computing Machinery, 2019.

[19] P. Testolina, F. Barbato, U. Michieli, M. Giordani, P. Zanuttigh, and M. Zorzi, "SELMA: SEmantic Large-Scale Multimodal Acquisitions in Variable Weather, Daytime and Viewpoints," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 7012–7024, 2023.

[20] Z. Song, Z. He, X. Li, Q. Ma, R. Ming, Z. Mao, H. Pei, L. Peng, J. Hu, D. Yao, *et al.*, "Synthetic Datasets for Autonomous Driving: A Survey," *arXiv preprint arXiv:2304.12205*, 2023.

[21] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017.

[22] J. Deschaud, "KITTI-CARLA: a kitti-like dataset generated by CARLA simulator," *CoRR*, vol. abs/2109.00892, 2021.

[23] M. Lyssenko, C. Gladisch, C. Heinzemann, M. Woehrle, and R. Triebel, "Instance Segmentation in CARLA: Methodology and Analysis for Pedestrian-Oriented Synthetic Data Generation in Crowded Scenes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 988–996, October 2021.

[24] J. He, K. Yang, and H. Chen, "6G Cellular Networks and Connected Autonomous Vehicles," *IEEE Network*, vol. 35, pp. 255–261, July 2021.

[25] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," *IEEE Commun. Mag.*, vol. 58, pp. 55–61, March 2020.

[26] X. Qiao, Y. Huang, S. Dustdar, and J. Chen, "6G Vision: An AI-Driven Decentralized Network and Service Architecture," *IEEE Internet Computing*, vol. 24, no. 4, pp. 33–40, 2020.

[27] X. Huang, G. Mei, J. Zhang, and R. Abbas, "A comprehensive survey on point cloud registration," 2021.

[28] D. Feldman, M. Schmidt, and C. Sohler, "Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering," *SIAM Journal on Computing*, vol. 49, no. 3, pp. 601–657, 2020.

[29] D. Xu and Y. Tian, “A comprehensive survey of clustering algorithms,” *Annals of Data Science*, vol. 2, pp. 165–193, 2015.

[30] S. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[31] D. Arthur and S. Vassilvitskii, “K-Means++: The Advantages of Careful Seeding,” vol. 8, pp. 1027–1035, 01 2007.

[32] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[33] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, “DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN,” *ACM Trans. Database Syst.*, vol. 42, jul 2017.

[34] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, “An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC),” *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.

[35] A. Varischio, F. Mandruzzato, M. Bullo, M. Giordani, P. Testolina, and M. Zorzi, “Hybrid Point Cloud Semantic Compression for Automotive Sensors: A Performance Evaluation,” in *ICC 2021 - IEEE International Conference on Communications*, 2021.

[36] R. Padilla, S. L. Netto, and E. A. B. da Silva, “A Survey on Performance Metrics for Object-Detection Algorithms,” in *International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, 2020.

[37] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 12697–12705, 2019.

[38] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[39] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[40] M. Drago, T. Zugno, F. Mason, M. Giordani, M. Boban, and M. Zorzi, "Artificial Intelligence in Vehicular Wireless Networks: A Case Study Using Ns-3," in *Proceedings of the Workshop on Ns-3*, WNS3 '22, (New York, NY, USA), p. 112–119, Association for Computing Machinery, 2022.

[41] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 46–51, 2020.

Ringraziamenti

Vorrei in primis ringraziare il professor Marco Giordani, senza cui il mio percorso universitario sarebbe stato senza ombra di dubbio molto diverso, vorrei ringraziarla per la pazienza e per essere sempre stato disponibile quando avevo bisogno di una mano.

Vorrei ringraziare Paolo Testolina, senza cui questo lavoro non sarebbe stato possibile, vorrei ringraziarti per essere sempre stato pronto ad accogliere ogni dubbio e ogni esitazione, per il tempo che abbiamo speso a guardare codice di dubbia correttezza e per aver sempre cercato di tirare fuori il meglio.

Vorrei ringraziare i miei genitori, Marina Allegro e Luca Bragato, senza cui sicuramente questa tesi non sarebbe stata scritta, ho sempre apprezzato il vostro modo di lasciarmi serenamente sbattere la testa, facendomi capire che eravate sempre lì qualora ne avessi avuto bisogno.

Vorrei ringraziare Alessia Ortile, Annamaria Pavan, Linda Crivellari, Luca Moretti, Mattia Peron, Marta Fantin e Samuele Vanini, senza cui questa tesi sarebbe stata un'esperienza meno felice, volevo ringraziarvi per far parte della nostra grande larga famiglia, ma soprattutto per aver capito che le difficoltà sembrano meno insuperabili quando ci si lamenta insieme.

Ma vorrei ringraziare anche personalmente Alessia Ortile, senza cui sarei diventato matto scrivendo questa tesi, anche se te ne vai lontana spero che prima o poi anche tu possa tornare in questo posto meraviglioso, grazie per aver condiviso con me quello scosceso ripido sentiero.

Ma vorrei ringraziare anche personalmente Annamaria Pavan, senza cui

questa esperienza sarebbe stata piena di domande e priva di risposte, grazie per avere la straordinaria abilità di riportarmi con i piedi per terra, grazie per riuscire a capire profondamente le mie preoccupazioni e grazie per cercare di tirare fuori sempre il bello dalle cose.

Ma vorrei ringraziare anche personalmente Linda Crivellari, senza cui mi sarei bruciato le ali cercando di raggiungere il sole, grazie per capire quando ho bisogno di dire di no e per aver capito come dirmi che devo dire di no, grazie per esserci sempre quando ho bisogno di condividere qualcosa che non dovrei dire a nessuno e grazie per essere sempre lì quando abbiamo bisogno di ripetere mille volte le stesse cose e stupirci ogni volta.

Ma vorrei ringraziare anche personalmente Luca Moretti, senza cui mi sarei perso un grosso pezzo di questo viaggio, volevo ringraziarti per sapere come trascinarci tutti all'avventura, per essere sempre disponibile se ce n'è bisogno, per riuscire a capire i problemi e per saperli condividere.

Vorrei ringraziare Gianmaria Ventura e Tommaso Lotta, senza cui avrei imparato molto meno dall'università, grazie per aver ascoltato i vari deliri di questi anni, grazie per aver lavorato con me e per aver premuto il freno quando ce n'era bisogno.

Vorrei ringraziare Lucia Borin, senza cui mi sarei annoiato moltissimo, grazie per essere quella bussola che periodicamente dimentico di consultare, anche se sa sempre dove è il nord.

Vorrei ringraziare il gruppo scout Lissaro 1, ma in particolare miei esploratori, senza cui oggi sarei radicalmente una persona diversa, grazie per insegnarmi ogni giorno qualcosa di nuovo, grazie per riempirmi di speranza, grazie per motivarmi. E, fra tutti voi, vorrei fare un ringraziamento in particolare. Grazie perché siamo dello stesso sangue, tu ed io, fratello mio.