Karlsruhe Institute of Technology
Fakultät für Elektrotechnik

Institut für Technik der
Informationsverarbeitung (ITIV)

TalkyCars: A Distributed Software Platform for Cooperative Perception among Connected Autonomous Vehicles based on Cellular-V2X Communication

Master's Thesis by

B. Sc. Ferdinand Mütsch

January 31, 2020

Head of Institute: Prof. Dr.-Ing. Dr. h.c. J. Becker
Prof. Dr.-Ing. Eric Sax
Prof. Dr. rer. nat W. Stork

Author: B. Sc. Ferdinand Mütsch

Supervisor: M. Sc. Martin Böhme
M. Sc. Marco Stang

Abstract

Autonomous vehicles are required to operate among highly mixed traffic during their early market-introduction phase, solely relying on local sensory with limited range. Exhaustively comprehending and navigating complex urban environments is potentially not feasible with sufficient reliability using the aforesaid approach. Addressing this challenge, intelligent vehicles can virtually increase their perception range beyond their line of sight by utilizing Vehicle-to-Everything (V2X) communication with surrounding traffic participants to perform cooperative perception. Since existing solutions face a variety of limitations, including lack of comprehensiveness, universality and scalability, this thesis aims to conceptualize, implement and evaluate an end-to-end cooperative perception system using novel techniques. A comprehensive yet extensible modeling approach for dynamic traffic scenes is proposed first, which is based on probabilistic entity-relationship models, accounts for uncertain environments and combines low-level attributes with high-level relational- and semantic knowledge in a generic way. Second, the design of a holistic, distributed software architecture based on edge computing principles is proposed as a foundation for multi-vehicle high-level sensor fusion. In contrast to most existing approaches, the presented solution is designed to rely on Cellular-V2X communication in 5G networks and employs geographically distributed fusion nodes as part of a client-server configuration. A modular proof-of-concept implementation is evaluated in different simulated scenarios to assess the system's performance both qualitatively and quantitatively. Experimental results show that the proposed system scales adequately to meet certain minimum requirements and yields an average improvement in overall perception quality of approximately 27 %.

Zusammenfassung

Autonome Fahrzeuge müssen besonders während ihrer frühen Markteinführungsphase in der Lage sein, sich in homogenem Mischverkehr zurecht zu finden. Dabei reicht es möglicherweise nicht aus, sich lediglich auf die lokale Sensorik mit eingeschränkter Reichweite zu verlassen, um komplexe, innerstädtische Verkehrssituationen zuverlässig wahrzunehmen. Stattdessen können intelligente Fahrzeuge ihre Sensorreichtweite durch den Einsatz von Vehicle-to-Everything (V2X) Kommunikation und Cooperative Perception virtuell über ihren ursprünglichen Horizont hinaus erweitern. Bisherige Ansätze gehen jedoch mit einer Reihe von Einschränkungen einher, da sie oftmals wenig holistisch, nicht ausreichend allgemeingültig oder schwer skalierbar sind. Daher hat diese Masterarbeit das Ziel, ein umfangreiches Cooperative Perception System auf Basis modernster Techniken zu entwerfen, implementieren und evaluieren. Dazu wird zunächst ein umfassender, aber dennoch leicht erweiterbarer Modellierungsansatz für dynamische Verkehrssituationen vorgestellt, der mithilfe probabilistischer Entity-Relationship Modelle ungewisse Umgebungenswahrnehmungen unterstützt und einfache Attribute mit abstrakteren, relationalen und semantischen Informationen auf möglichst generische Weise kombiniert. Anschließend wird der Entwurf einer ganzheitlichen, verteilten Software Architektur auf Basis von Edge Computing-Prinzipien als Grundlage für fahrzeugübergreifende Sensorfusion vorgestellt. Im Gegensatz zu den meisten bestehenden Ansätzen setzt unsere Lösung auf Cellular-V2X Kommunikation in 5G-Netzen und verwendet geographisch verteilte Rechenknoten. Eine modulare Proof-of-Concept Implementierung wird in verschiedenen simulierten Szenarien evaluiert, um das System sowohl qualitativ als auch quantitativ zu bewerten. Entsprechende Experimente zeigen, dass das vorgestellte System ausreichend gut skaliert, um bestimmte Mindestanforderungen zu erfüllen und erzielt eine durchschnittliche Verbesserung der Wahrnehmungsqualität von ca. 27 %.

Erklärung

Ich versichere hiermit, dass ich meine Masterarbeit selbständig und unter Beachtung der Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) in der aktuellen Fassung angefertigt habe.
Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, den 31. Januar 2020

Ferdinand Mütsch

Contents

Abbreviations

3GPP	3rd Generation Partnership Project
AD	Autonomous Driving
ADAS	Advanced Driver Assistance System
API	Application Programming Interface
BEV	Battery Electric Vehicle
C-V2X	Cellular Vehicle-to-Everything
CAM	Cooperative Awareness Message
CEN	European Committee for Standardization
CP	Cooperative Perception
CPM	Cooperative Perception Message
DSRC	Dedicated Short-Range Communications
ER	Entity Relationship
ETSI	European Telecommunications Standards Institute
FCEV	Fuel Cell Electric Vehicle
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IoT	Internet of Things
ITS	Intelligent Transportation Systems
LiDAR	Light Detection and Ranging
LOS	Line of Sight
LTE	Long Term Evolution
NLOS	Non Line of Sight
OBU	On-Board Unit
P2P	Peer-to-Peer
PER	Probabilistic Entity Relationship
PHEV	Plug-In Hybrid Vehicle
QoS	Quality of Service
ROP	Reliability of Perception
RSU	Road-Side Unit
SLAM	Simultaneous Localization and Mapping
V2C	Vehicle-to-Cloud
V2G	Vehicle-to-Grid
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle

V2X	Vehicle-to-Everything
VANET	Vehicular Ad-Hoc Network

Chapter 1

Introduction

This first chapter introduces the interested reader to the subject area of this thesis and its main purposes and demonstrates the demand for research on the covered topics.

1.1 Motivation

Public and academic interest in Autonomous Driving (AD) has grown tremendously over the past decade. As a technology that holds great potential to significantly increase security, efficiency and driver's comfort and to reduce the number of casualties on the road by up to 90 % [Mar17a] it is an inevitable step towards an upcoming revolution in transportation. Although it is still hard to predict when fully self-driving cars will be publicly available [Fro18], technological progress is being achieved at an increasingly rapid pace. Primarily enabled through recent advances in Artificial Intelligence, computation hardware and optical sensor technology, AD systems are continuously becoming more robust and accurate.

However, perception accuracy of today's Advanced Driver Assistance Systems (ADASs) is limited by the range of on-board sensory and a vehicle's line-of-sight (LOS). To be able to safely navigate through complex urban environments, an intelligent vehicle might additionally rely on external observations obtained by surrounding traffic participants, which it constantly exchanges information with through Vehicle-to-Everything (V2X) communication. This concept of combining sensor information across multiple agents to improve perception quality is referred to as Cooperative Perception (CP) and has proven beneficial to address the problem of limited perception and accuracy [CTYF19, HKS$^+$19]. Its presence is particularly expedient during the market introduction and early adoption phase of self-driving technology, when V2X-enabled cars will face mixed- or predominantly human-controlled traffic.

Cooperative Perception is holds enormous potential [GTW15] and research on related topic is gaining momentum recently [CTYF19, TSG19, CM17, BMW19]. Most current approaches, presented in chapter 3, rely on decentralized, ad-hoc communication and lack a uniform, yet flexible format for representing relevant aspects of a traffic scene. This entails a number of limitations, which are discussed in chapter 4. Moreover, to the best of my knowledge, no holistic CP system has been presented, yet.

Primary goal of this thesis is to conceptualize, implement and evaluate a comprehensive, end-to-end Cooperative Perception system using novel techniques. Emphasis is laid on the design of a reliable and scalable software architecture and an appropriate schema to model and exchange a shared environment state. Aspects covered in this context include, among others, the suitability and performance of communication via cellular networks, approaches to high-level fusion of time-delayed sensor data and the modeling of uncertain environments.

Chapter 2

Background

This chapter introduces to essential topics and concepts in the context of this work and provides the reader with background knowledge required to follow in later chapters.

2.1 Autonomous Driving

Academia and established industry leaders in automobile manufacturing are vigorously pushing research on autonomous driving technologies alongside emerging start-up companies, who try to enter the new market. The challenge of self-driving cars is believed to be solved within a few decades with high certainty [Fro18], although precise forecasts diverge. However, most experts agree that the benefits are enormous. Such include decreased risk of collisions and causalities, higher traffic efficiency, less occupied roads – leading to better environmental sustainability – and enhanced driver comfort. New business models – like robo-taxi- or car-sharing services – are likely to arise as transportation culture will undergo a shift from individually owned cars towards a sharing economy and *Mobility as a Service* concepts. Nonetheless, despite these advantages, AD is also accompanied by a number of challenges. Most importantly, government regulations and an appropriate legal framework are vital. Moreover, people are commonly concerned about the accompanying loss of jobs and the cultural changes in general [SS14].

2.1.1 Current Status

With reference to self-driving cars, a distinction is usually made between five different levels of autonomy [Kle18], presented in appendix section A.1.1. These levels are used to uniformly describe vehicles' capabilities and their degree of independence from a human driver with regard to the task of driving. This subsection outlines the status quo in autonomous driving research with regard to these five levels.

Many of the major car brands have level 2 vehicles in production today and some already offer models with experimental level 3 technology [Fro18]. One of the most famous examples is Tesla's AutoPilot[1], which is able to follow a route on the highway towards a given destination autonomously, while keeping and changing lanes on its own. According to [Fro18], *"China is expected to lead North America and Europe by the number of automated vehicles sold, whereas technology penetration wise, Europe is expected to lead the market for autonomous driving globally [by 2025]"*. By 2025, 2 million level 4 vehicles could be sold in Europe, while the first level 5 vehicles could reach production readiness by 2030 [McK19].
Market revenue for ADAS is expected to double by 2021 to reach $35 billion [McK19]. Accordingly, many OEMs, including *General Motors* and *Volkswagen*, invest in acquiring AD start-up companies to extend their technological know-how to gain competitive advantages [Kor19b, Kor19a]. In addition, big players from the tech industry and disruptive mobility suppliers push into the market with self-driving car fleets and shuttle services, including Uber[2], Lyft[3] and Waymo[4].
On the technological side, hardware manufacturers like Nvidia[5] and Qualcomm[6] invest in research on AD- and V2X-specific chips and machine learning hardware. Moreover, online education platforms like Coursera[7] and Udacity[8] offer specific courses on AD to target the increasing demand for experts on these subjects. With Baidu Apollo[9] and Autoware.AI[10] there are even comprehensive, end-to-end AD platforms available as open-source software to be used in simulation or installed on a real car.

2.1.2 Sensor Fusion

Additional sensors compared to non-autonomous cars are mainly required for two purposes: perception and localization. The former refers to the vehicle acquiring a detailed model of its surrounding, including type, position and speed of other traffic participants, traffic light state and more. The latter means to accurately find the vehicle's own position on a map. Current Level 2 vehicles already have a multitude of different sensors

[1]`https://www.tesla.com/autpilot`

[2]`https://www.uber.com`

[3]`https://self-driving.lyft.com/`

[4]`https://www.waymo.com/`

[5]`https://developer.nvidia.com/drive`

[6]`https://www.qualcomm.com/invention/5g/cellular-v2x`

[7]`https://www.coursera.org/lecture/machine-learning/autonomous-driving-zYS8T`

[8]`https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013`

[9]`https://github.com/ApolloAuto/apollo`

[10]`https://gitlab.com/autowarefoundation/autoware.ai`

and [Fro18] predict that future Level 5 cars might even have between 28 and 32 different sensors.

2.1.2.1 Sensors

For **localization**, mainly GPS (Global Positioning System) and IMU (Inertial Measurement Unit) sensors are used. The latter usually consists of a combination of accelerometers and gyroscopes and helps to locate the vehicle even when no GPS connection is available. Occasionally, laser sensors (LiDAR) and radar technology are used in addition for more accurate positioning. While some approaches tend to rely on detailed, high-definition maps for even more precise positioning, others oppose the necessity of a car to know its position at centimeter-level accuracy [FH19]. When using high-precision sensory, the sole localization problem is often extended to *Simultaneous Localization and Mapping* (SLAM).

For **perception**, most current approaches rely on (stereo) cameras, ultrasound sensors and radar. Some manufacturers consider LiDAR crucial in addition, while others, e.g. Tesla and Nissan [McK19], strictly oppose its use for perception or localization tasks. Comma.ai[11] even followed the approach of solely employing cameras for perception, arguing that, given the human example, decent driving performance can be achieved with only optical sensory.

2.1.2.2 Fusion

Sensor Fusion *“is the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually“* [Elm02]. This also includes data normalization and temporal alignment.

Chen et al. [CTYF19] differentiate between three levels (depicted in fig. 2.1) on which sensor fusion can happen, whereas the data subject to a fusion process is increasingly abstract at higher levels.

[11] `https://comma.ai`

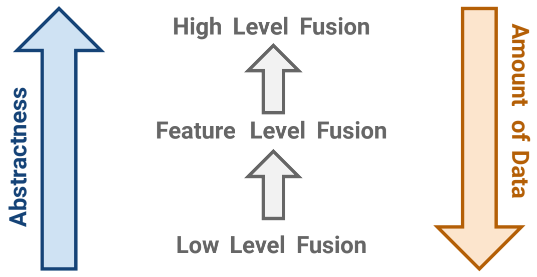


Figure 2.1: Levels of Sensor Fusion

- **Low Level Fusion:** Raw sensor data is subject to the fusion process. Input might be LiDAR point clouds, RGB camera images, etc. Commonly used algorithms are Kalman filters, Bayesian networks and, more recently, also Neural networks.

- **Feature Level Fusion:** Before fusing, certain features are extracted from the raw data. For instance, if some component within the AD stack is responsible for lane keeping, lane markings could be extracted from raw RGB images for this purpose, e.g. using a Canny filter. Input might either be raw sensor data or the outputs from a subsequent low-level fusion step.

- **High Level Fusion:** High-level fusion operates on the level of objects, which are extracted from sensor data. In the context of Cooperative Perception, these objects are usually other traffic participants with their respective properties. Input will usually be the outputs of some form of preceding low- or feature-level fusion.

As explained in greater detail in later chapters, this work will mostly deal with high-/object-level fusion.

2.1.3 Autonomous Driving Pipeline

As stated by [CML+18] and [FH19], there are two opposing sides from which the problem of autonomously driving a car could be approached. The first one is a **modular** system, in which a multitude of different components and sensors perform various kinds of tasks to create a model of the environment and use it for planning and control. This is the most widely used approach and subject of the majority of today's research on AD. However, especially recent advances in the field of deep-learning gave rise to a second paradigm: **end-to-end** techniques. Such attempt to *"train function approximators to map sensory input to control commands"* [CML+18].

Both strategies come with their own advantages and downsides, but since the modular approach is more transparent and well understood, it is the only one to be subject of further discussion.
Following a modular approach, the process from sensing the environment to controlling a self-driving car can be categorized into steps of a pipeline, that is repeatedly run.

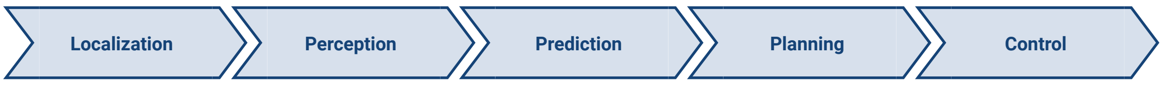


Figure 2.2: Autonomous Driving Pipeline

1. **Localization:** Often implemented as a combined *Simultaneous Localization and Mapping* (SLAM) problem, goal is to determine the vehicle's current position on a map with high precision. Common algorithms and sensors used in this step were described in section 2.1.2.

2. **Perception:** A crucial step towards AD is to perceive a vehicle's current environment, where perception is defined as the *"process of maintaining an internal description of the external environment"* [CD93]. That includes to recognize, classify and locate other traffic participants and their static and dynamic properties as well as any other surrounding obstacles. In this thesis, the perception step is of primary interest. Common algorithms and sensors used in this step were described in section 2.1.2.

3. **Prediction:** Prediction builds on the outcome of the preceding perception step and aims to estimate a future state $\hat{\theta}_{t+1}$ of the vehicle's surrounding environment, given an observation θ_t at present time. For instance, the future trajectory of a nearby car might be approximated. The problem of **tracking** objects over multiple frames / multiple repeated executions of the AD pipeline can also be considered part of the prediction step.

4. **Planning:** Given the vehicle's own current position as well as estimations for the future state of all other nearby traffic participants, the planning step aims to find an appropriate path that satisfies certain requirements and minimizes given cost metrics. For instance, given a global target position, a collision-free trajectory might be found that is physically feasible, compliant with traffic rules, minimizes *jerk* (the accumulated magnitude of the acceleration change [PCY+16]), and maximizes the average distance to all obstacles. Planning can be divided into the sub problems of **(1) Routing**, **(2) Behavior Planning** (e.g. choose an action

$a \in \{\text{“}keep\_lane\text{“}, \text{“}change\_lane\text{“}, \text{“}stop\text{“}, \text{“}accelerate\text{“}, ...\}$ and **(3) Motion Planning**. (3) again, usually is solved in two steps, namely **(3.1) Path Planning** and **(3.2) Trajectory Planning**. Problem (3.1) is considered PSPACE-complete [PCY+16]. Commonly used algorithms include, but are not limited to A* and RRT*.

5. **Control:** Eventually, planning output needs to be translated into actual brake-throttle- and steering commands to physically control the vehicle through its actuators.

2.2 Vehicle-to-X Communication

Vehicle-to-X, or Vehicle-to-Everything, communication generally describes the Internet Of Things (IoT) approach of having (autonomous) vehicles exchange information with other actors in their local environment through messaging.

2.2.1 Application Types

A V2X communication system can have different constellations, depending on what participants are involved.

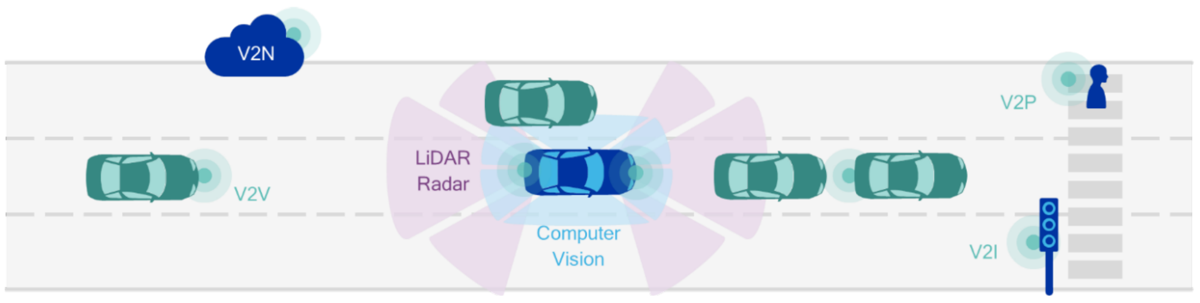


Figure 2.3: Types of Vehicle-to-Everything Applications [5G 16]

- **Vehicle-to-Vehicle (V2V):** The ability of cars to communicate with each other. This is among the most common instantiations of Vehicle-to-Everything communication.
- **Vehicle-to-Infrastructure (V2I):** The ability of cars to communicate with any type of traffic infrastructure. Most commonly, vehicles bilaterally exchange information with traffic lights to optimize traffic flow. Further examples include automated fare collection or emergency vehicles path cleaning.
- **Vehicle-to-Pedestrian (V2P):** The ability of cars to indirectly communicate with nearby pedestrians, mainly used to prevent collision. Pedestrians need to

be equipped with smartphones or wearable devices to participate in the communication.

- **Vehicle-to-Network (V2N):** The ability of cars to communicate with network services over a macro cell, as opposed to communication to smaller RSUs in the case of V2I [AzPA+19].
- **Vehicle-to-Grid (V2G):** The ability of cars to communicate with entities of the electrical power grid. With this approach, *"[...] plug-in electric vehicles, such as battery electric vehicles (BEV), plug-in hybrids (PHEV) or hydrogen fuel cell electric vehicles (FCEV), communicate with the power grid to sell demand response services by either returning electricity to the grid or by throttling their charging rate."* [Wik19e]
- **Vehicle-to-Cloud (V2C):** The ability of cars to communicate with all kinds of cloud services, e.g. to get real-time traffic information, find parking spots, receive over-the-air software updates from its vendor or consume multi-media entertainment.

Especially V2I and V2G patterns will likely emerge as "Smart Cities" establish in the near future. Even today, the first V2I solutions are already in place. For instance, the U.S. city of Tampa, Florida has installed the *Connected Vehicle Pilot* system, in which cars communicate with traffic lights and other RSUs to optimize traffic flow [Tam18]. Another use case for V2I today is ambulances communicating with traffic lights to enable a green wave so they reach their destination quicker [Isr19]. Moreover, the German Federal Highway Research Institute (Bundesanstalt für Straßenwesen) provides a central *Marketplace for Mobility Data* (MDM)[12] as a commercial web service to bring together potential consumers and producers of different kinds of mobility- and traffic-related data. Such include measurements from traffic and environment detectors, parking information, information on road works, hazards and incident alerts, petrol station prices and more.

Given the above classification it is worth noting that, in a strict sense, V2V, V2I and V2P only describe direct communication between the participants (e.g. car and car or car and RSU). If there is an intermediary, like a cell tower, involved, proper terminology is to speak of, for instance, Vehicle-to-Network-to-Vehicle (**V2N2V**) [5G 19b]. However, for the sake of simplicity, terminology in this thesis will neglect the presence of an intermediary, i.e. V2N2V (and similar) will be named just V2V.

This thesis mainly addresses V2V and V2I use cases.

[12]`https://www.mdm-portal.de/?lang=en`

2.2.2 Communication

There are two types of V2X communication technology depending on the underlying technology being used, namely *Dedicated Short-Range Communications* (DSRC) and Cellular-V2X (C-V2X)-based approaches. Each of them imply different communication patterns. Today, DSRC is more common. It usually relies on WiFi-based communication using the IEEE 802.11p standard and implies *Vehicular Ad-Hoc Networks* (VANETs) as communication topology. In the case of VANETs, vehicles and infrastructure devices (or road-units (RSUs)) in range form pairwise connections among each other and build up a peer-to-peer (P2P) network. Besides direct message exchange, VANETs usually also support multi-hop communication to reach out to further distant actors beyond WiFi range and line-of-sight. The European version of DSRC, standardized by CEN[13], is also referred to as ITS-G5 to avoid confusion.

In 2016, a first specification of C-V2X technology using Long Term Evolution (LTE) networks was published by 3GPP[14]. Especially with the upcoming establishment of 5G networks, whose characteristics are shown in section 2.5, C-V2X is becoming increasingly attractive as dramatically improved latency and throughput allow for high-performance applications. With C-V2X, both P2P-based- as well as centralized, wide-area communication are possible.

2.3 Cooperative Perception

2.3.1 Theory

Although perception accuracy and reliability have greatly improved over time, current systems still fail to completely comprehend complex situations occasionally. This may lead to incorrect decisions and potentially to collisions. Consequently, relying on local sensors only is insufficient under certain circumstances, especially in situations with very limited line-of-sight (LOS).

Using V2X communication, next generation vehicles might be able to extend their perception range vastly [CTYF19, HKS+19]. More precisely, they might be enabled to exchange information about their own state in combination with sensor data or a higher-level local environment model. Connected cars could become an additional, virtual sensor to each other. Accordingly, cooperative (or collective) perception can be viewed as a sensor fusion problem, whereas a "cooperative" sensor network is characterized as such that *"[...] uses the information provided by two independent sensors to derive information that would not*

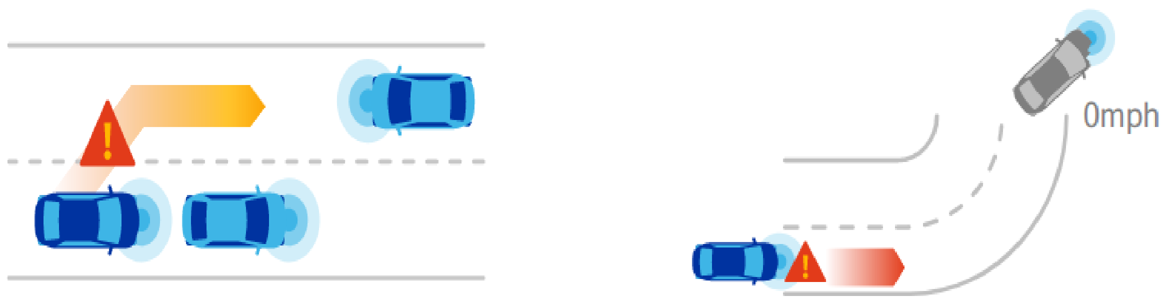
[13]`https://www.cen.eu`

[14]`https://www.3gpp.org`

be available from the single sensors.“ [Elm02]. The different fusion levels introduced in section 2.1.2 apply analogously with respect to the data being shared among CP-enabled vehicles (e.g. raw sensor measurements or high-level objects).
However, as already found by [GTW15], a drawback of CP – as with any other system relying on the presence of a network – is the network effect: a CP system is only useful once the number of participants exceeds a certain critical mass. Accordingly, for CP to succeed, it is crucial to follow a rather aggressive market penetration strategy during its introduction. Preferably, different OEMs would collaborate to build one unified system.

2.3.2 Use Cases

Cooperative Perception is expected to bring two essential improvements. First, vehicles' field-of-view is extended virtually (**Non Line-of-Sight Sensing [NLOS]**). Second, **confidence** for observations within an area, that is overlapped by the perception range of two or more connected cars, can be improved through “voting“.



(a) Opposing traffic out of sight (b) Vehicle behind a curve

Figure 2.4: Exemplary Non Line-of-Sight Scenes [Qua17]

Figure 2.4 depicts two traffic situations in which CP can greatly help to improve safety. In fig. 2.4a, the leftmost vehicle is about to overtake, but can not see the opposing traffic, because its sight is restricted by the vehicle in front. However, since both other cars are broadcasting their own state and their perceptions, it is virtually moved into range of sight. Similarly, in fig. 2.4b, the blue car can only recognize the stopped car once it has already passed the curve and might have to brake sharply without CP.
Since Cooperative Perception can be seen as a subset of V2X in general, all known benefits with V2X systems can be incorporated into a CP system as well. Those include additional safety through **intent announcements** and **automated emergency brake lights** as well as traffic flow optimization through **situation-aware signal phase timing** [5G 19a].

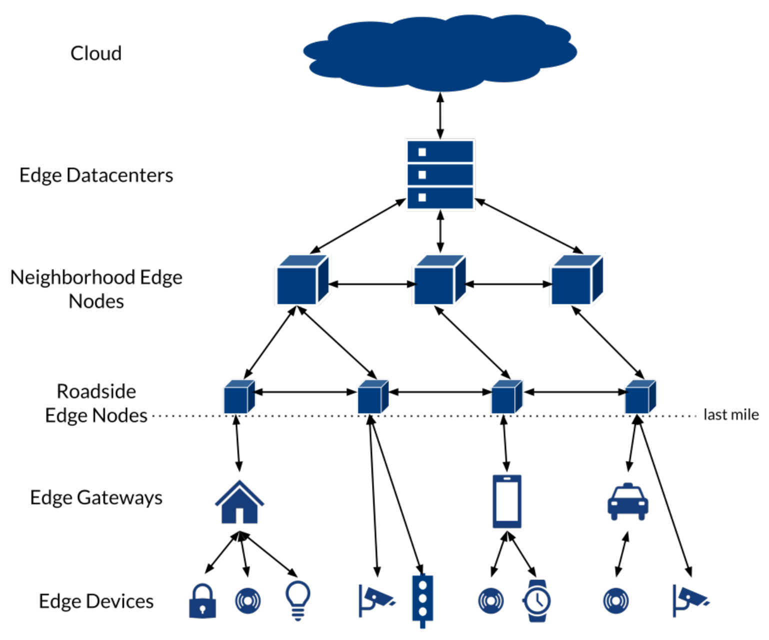


Figure 2.5: Architecture of Edge Computing [BSKH19]

2.4 Edge Computing

Edge computing is an architecture design pattern for distributed software applications and schematically depicted in fig. 2.5. *"In general, [it] [...] is the process of performing computing tasks physically close to target devices, rather than in the cloud or on the device itself"* [BSKH19]. Usually, one or more additional layers of computation devices (edge nodes, edge gateways) are introduced as intermediate "hops" between end-device and the cloud. This is especially beneficial in IoT contexts, where devices are comparatively weak in terms of computational capabilities, while the amounts of gathered data can quickly become enormous. Accordingly, on the one hand, high load is taken from those low-power devices and, on the other hand, latency between device and analytics server is kept small. In the context of Cooperative Perception, low latency is especially crucial, so edge computing appears to be a promising pattern.

Besides having stronger hardware (and therefore higher **compute capacity**) for data processing tasks compared to the end-devices themselves and better **latency** compared to using cloud infrastructure, edge computing comes with additional advantages. Through a higher degree of distribution **reliability**, **scalability** and **robustness** can be improved. Moreover, by employing edge servers, **costs** can be reduced and **security** can be increased, especially when dealing with privacy-sensitive applications and data.

Recently, another term for a similar concept has established: **fog computing**. While boundaries between edge- and fog computing are blurry, one could argue that fog nodes could make up an additional layer of abstraction and are placed between edge- and cloud nodes. In the example of a large IoT-enabled factory, edge nodes could exist within the local area network of each shop floor, while one or more fog nodes are located in a company-internal micro data center. Both of them would usually have the task to analyze and process data from the previous step to pre-process, filter and summarize it before it eventually gets send into the cloud.

In the context if this thesis, an additional layer of indirection is not considered necessarily and, therefore, the terms are used interchangeably for the sake of simplicity.

2.5 5G Cellular Networks

5G stands for the fifth generation cellular network standard and is the successor of 4G, or LTE. It may be operated on a variety of different spectrums, ranging from low-band (~600 Mhz) over mid-band (2.4 to 4.2 GHz) to millimeter waves in the high-band spectrum ranging from 24 to 72 GHz. Which spectrum is used depends mainly on the carrier and has direct influence on communication range and speed. While millimeter wave frequencies will mainly be used in North America, Europe will rely on the low- and mid-range bands.

In networks based on the most widely used mid-band spectrum, average throughput is between 100 and 400 $\frac{\text{Mbit}}{\text{s}}$, while it can increase up to 2 $\frac{\text{Gbit}}{\text{s}}$ with millimeter waves [Wik19a].

Typical round-trip times (RTT) were measured to range from 25-35 ms and might be improved to 10-20 ms when employing an *Edge Node* (see section 2.4) close to a cell tower [Wik19a]. Under lab conditions, even less than 1 ms latencies were observed.

Like 4G, 5G is based on network cells between which moving mobile clients are handed over. Generally, network cells are smaller with 5G than with 4G, especially in densely populated areas. Depending on which frequency band is used, cell towers may need to be placed every few hundred meters to achieve full coverage.

2.6 Geo Tiling

Geo tiling, or *hierarchical binning*, is a strategy to represent, uniquely identify and index geospatial data. The idea is to *"store a geo-database such that data for a specific location can be retrieved quickly, by dividing the data up by location, partitioning the world into tiles"* [Ope18].

2.6.1 QuadKeys

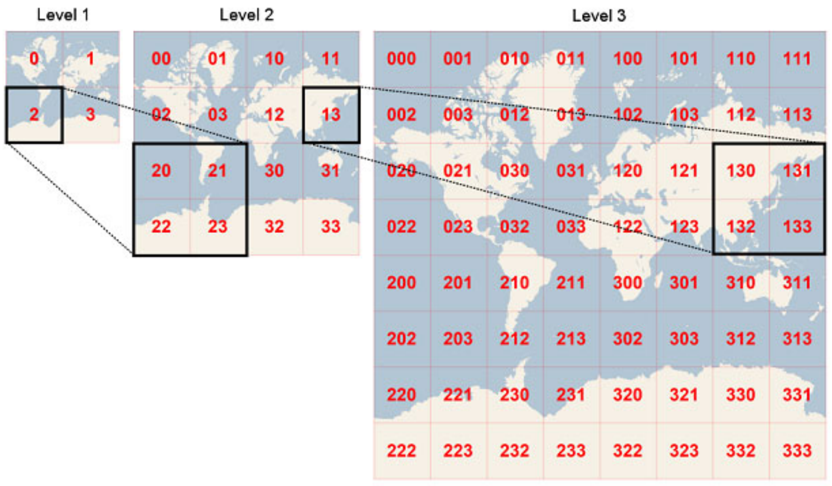


Figure 2.6: Geo Tiling with QuadKeys [Ope18]

One implementation of geo tiling is *QuadKeys*, proposed by [Sch18]. Its idea is to recursively split the two-dimensional *Mercator projection* of the world map into four square tiles. Assuming an earth circumference of 40 000 km, every tile would be approximately 20 000 km by 20 000 km in size at the first level. Each of these tiles is split into four tiles again, while their size is halved in every iteration. Continuing this way, arbitrary precision can be reached in theory.
As depicted in fig. 2.6, every tile is uniquely identifiable by a number (or a text string) when enumerated recursively. The length of that string, and therefore the maximum spatial precision, is only limited by the size of the data types used during its calculation. When using 64-bit float variables, the maximum level is 54. At level 30, for instance, ground resolution at the equator is already 3.7 cm^2, while at level 54 it is $2.22*10^{-9}$ cm^2.

In the course of this thesis, QuadKeys are used to uniquely and uniformly reference geographical locations.

Chapter 3

Related Work

This chapter aims to provide the reader with an overview of current research and state-of-the-art in the field of cellular V2X communication and cooperative perception.

In accordance with the multiple goals of this thesis introduced in chapter 1 and chapter 4, related work can, in the broadest sense, be separated into three sections. On the one hand, relevant publications about **environment modeling, traffic scene representation and message exchange formats** for cooperative perception are examined in section 3.1. Secondly, an overview of existing **CP systems** is presented and different approaches are discussed in section 3.2. Finally, related work on **cellular-based V2X** is presented.

3.1 Environment Modeling & State Representation

An essential requirement for high-level CP is to have a uniform way to first model the current environment state and second represent that model in form of exchangeable data structures. It is worth noting that this is not necessarily required in the case of low- or feature level CP (see section 2.1.2), where either raw sensor readings or only basic information is shared.
An appropriate state representation should, at a minimum, include information about position and dynamics of the sender vehicle and all other surrounding traffic participants. For the former, the European Telecommunications Standards Institute (ETSI) has defined a standard for so called **Cooperative Awareness Messages** (CAM) [Eur11], which is used by Rauch et al. [RKD11]. It includes, among others, the sender vehicle's type, its dimensions, position, heading, speed and acceleration as well as respective confidences. To share information about surrounding obstacles in addition, the simplest way is to send **object lists** that include such. For this purpose, Rauch et al. [RKD11] defined the **Cooperative Perception Message** (CPM) in 2011. Since 2017, the ETSI is working

on a similar specification with the same name [Eur19]. It includes an object list of up to 255 traffic participants [TSG19] with attributes similar to those included in CAMs. However, none of both specifications is publicly accessible, yet. Therefore, they can not serve as a basis for this thesis.

Despite pure object lists, another way to share the perception of one's local environment is to model it in the form of **occupancy grids** or *driveable area* maps [Pie13]. However, to the best of my knowledge, no CP solution exists that relies on exchanging occupancy grids.

Kohlhaas et al. [KBSZ14] first introduced the concept of **semantic scene representation** (or *semantic state representation*), which is further advanced by Wolf et al. [WKW$^+$18] to *"combine low level attributes with high level relational knowledge in a generic way"*. They explain several advantages of including semantic, relational information among entities over exchanging plain object lists or occupancy grids. In [PAKZ18], semantic modeling is picked up again and combined with the idea to use **Probabilistic Entity Relationship** models for state representation under uncertainty. For none of these three approaches did the authors share a complete meta-model or ontology for traffic scenes.

A few further approaches to modeling and state representation in the context of automated driving are mentioned in appendix section A.2.1.

In summary, the previously mentioned takes on modeling and representation of road scenes constitute great building blocks for a CP system. This work aims to combine some of their conceptions to come up with a holistic way to (1) model a traffic scene by all relevant aspects, (2) represent it in an appropriate format and (3) define in what form to efficiently transfer it over the wire (or wirelessly).

3.2 Cooperative Perception

Different approaches to design a cooperative perception system have already been published in the past, each of them placing a different focus.

An early take in the field of V2X communication and – in the broadest sense – also cooperative perception was presented by Olaverri-Monreal et al. [OMGF$^+$10]. Based on VANETs, they propose a *see-through system* that allows a driver to virtually perceive the road in front of a large obstacle, e.g. a truck, blocking her sight. To achieve this, DSRC is used to exchange raw camera images between connected participants. However, this approach is somewhat special in a way that it define only an driver assistance system (or ADAS) rather than a CP-system for driverless cars in the sense of this work.

Secondly, as part of the Ko-PER research project[1] the authors of [RKD11] propose a system architecture based on **track-to-track fusion**, in which local- and global fusion is performed in separate steps to solve the problem of correlated data. Communication is based on IEEE 802.11p and utilizes CAMs and self-defined CPMs, for which they distinguish further between iCPMs (infrastructure) and vCPMs (vehicle), depending on the type of sender. Primary objective of their studies is to get detailed insights about **latency and transmission range** in CP scenarios. Eventually, the system is evaluated in both a laboratory and a real-world setup involving two cars. While their findings are essential for understanding performance-relevant parameters in V2X setups, the evaluation might have been conducted for more realistic scenarios with many traffic participants in addition.
Complementing their previous work in the field of CP, the authors of [RKRD12] intensively study methods for **temporal and spatial alignment** of cooperative perception messages. A motion model is used to predict observations to current time on the receiver side. In addition, they compare two types of possible transformations to accurately estimate an object's spatial position as combination of local- and received observations. While [RKD11] is concerned with communication aspects, [RKRD12] mainly addresses fusion. Findings from both works are well qualified to be incorporated into a more comprehensive solution.

While the above studies focus on techniques and parameters within the process of CP itself, [LKC+13] investigates the impact of CP on **motion planning** tasks. It is suggested to use local observations for short-term navigation and combined, global observations for longer-term planning. Therefor, a cost map is continuously re-constructed, that additionally includes weights corresponding to the underlying observations' **reliability of perception (ROP)**. Instead of IEEE 802.11p, their connected vehicles use conventional WiFi (802.11n) for P2P communication. Their custom state model does not include any additional information except the probability (confidence) for the quadruple state $X \in \{\chi_{correct}, \chi_{opposite}, \chi_{offroad}, \chi_{roadobst}\}$ of a certain location being driveable or not. As a result of evaluating planning quality with respect to trajectory smoothness and accumulated cost they found CP to be beneficial.
Another evidence for the positive effect of CP on path planning is given by the authors' subsequent publication [KCQ+13]. This complementary work mainly addresses the problems of **vehicle identification**, **delay compensation** and merging **occupancy grid maps**, which are used as an environment model. In contrast to previously mentioned systems, most of which either use high- or feature level fusion, [KCQ+13] is based on low-level fusion, i.e. raw sensor observations (LiDAR and camera) are exchanged.

The system design proposed by Calvo et al. [CM17] is the most comprehensive one in this

[1] `http://ko-fas.de/41-0-Ko-PER---Kooperative-Perzeption.html`

collection, as it presents a 3+1 level CP architecture to be used as an *obstacle avoidance framework*. DSRC-based V2V communication with IEEE 802.11p is used to exchange participants' immediate local surrounding. Long-range, **cellular**, LTE-based V2I communication facilitates the exchange of more global observations to be used for high-level tasks like traffic re-routing. In addition, their concept also includes I2I communication between RSUs. In comparison to previously mentioned approaches, which mostly cover certain CP-related aspect, the present paper focuses on conceptually outlining a **system architecture**. However, neither an environment model nor a concrete state representation or message format are presented and it is not made clear what kinds of information are exchanged at all.

[CTYF19] is the most recent publication in the field of CP and relies on low-level fusion of 3D LiDAR point clouds. After motivating their approach and outlining several accompanying challenges, the author present their DSRC-based system design. Since raw sensor data is sent, the design does not include the specification of an environment model. Instead, particular emphasis is laid on (1) the **fusion of point clouds**, (2) **deep-learning-based object detection** and (3) **optimization of network utilization**, i.a. by determining a certain **region of interest** (ROI) for every frame. As a consequence of their evaluation, the authors find that the effective sensing area can successfully be expanded with CP to capture previously unknown obstacles.

One of the most comprehensive and closely related projects is Ko-HAF [HKS+19], sponsored by the German Federal Ministry of Education and Research. As a holistic, V2V solution it comprises, among others, message- and format specification and the definition and implementation of an overall system architecture. As a *"standard for the exchange of information between vehicle sensor and back-end solutions"* they present SENSORIS as an extension to the Sensor Data Ingestion Interface (SDII) format specification by HERE Maps[2]. The presented system architecture involves their so called *Safety Server* as a **central software component** to be responsible for data consolidation, fusion and redistribution. Participant vehicles communicate with it via cellular (LTE) network and exchange binary serialized messages via HTTP and MQTT. However, while structurally and technically very similar, their approach is not meant to be used for CP in the sense of this work. Instead of aspiring high-precision collaborative awareness and understanding of dynamic traffic scenes for driverless cars, they rather attempt to collaboratively **build and update a global, "learning" map** for ADAS. In other words, they aim for collaborative SLAM, to some extent. Accordingly, the system's focus is on exchanging static road information – especially road signs and lane topology and markings – on the

[2]`https://developer.here.com/olp/documentation/sdii-data-spec/topics/introduction.html`

one hand and traffic events and incidents (like construction zones, congestions, stopped cars, etc.) with temporal validity on the other. It is not designed for high dynamicity and does not support to exchange information about surrounding traffic participants.

3.3 Cellular V2X Communication

With respect to cellular V2X communication – as opposed to DSRC-based solutions – recent standards are being developed by 3GPP most notably with the publication of [3GP19].
In addition, [Qua18] conducted comprehensive experiments to measure the performance of LTE and 5G networks with respect to Cooperative Perception and found 5G to be well suited for these use cases.
This is complemented by the results of [5G 16], whose authors found cellular, 5G-based communication to be superior over DSRC / ITS-G5 for CP tasks.

3.4 Summary

Great research has already been contributed to the fields of traffic scene modeling, environment state representation for autonomous driving, cooperative perception and (cellular-) vehicle-to-everything communication. However, several limitations exist with current approaches, that are discussed in greater detail in chapter 4. Most notably, the majority of existing CP solutions in literature rely on DSRC-based VANET communication topologies and on-vehicle data fusion, though the rise of high-performance cellular networks allows for entirely new concepts and system design patterns. Accordingly, this work builds up on previous findings and investigates new possibilities enabled through technological advances to design a holistic, modern system.

Chapter 4

Problem Analysis

This chapter first outlines limitations of current approaches in the field of cooperative perception and motivates this work's contributions to overcome them. Second, goals and conceptual requirements are presented as a guideline for later system design. Finally, it is made clear which aspects are in or out of scope of this thesis.

4.1 Limitations of Prior Work

Existing work in the field of cooperative perception, as presented in chapter 3, faces several limitations.

One of them is a **lack of comprehensiveness**. While the shown studies each focus on certain aspects of a CP system, to the best of my knowledge, no solution was presented that takes both a macro and micro perspective, i.e. that is holistic as well as concerned with details about all individual aspects and parts. Also, not all previous projects provide an actual implementation of their proposal or conduct realistic simulations.

Moreover, **no standardized, uniform, yet expressive environment model** and state representation for traffic scenes exists in literature, yet. While the efforts taken by [Eur19] to specify a standard for CPMs are promising, at this point, there is nothing like that available to be used in CP systems. Current models are either incomplete, proprietary and non-transparent or not suitable for interoperability between heterogeneous systems.

Another significant issue arises from limitations regarding **scalability of VANET-based cooperative perception systems**.
One essential challenge is the limited **throughput** and **range** of DSRC networks, which are usually based on IEEE 802.11p technology. [5G 18] found DSRC to be 90 % reliable at 675 m distance between sender and receiver in line-of-sight scenarios and 375 m in

non-line-of-sight situations. However, Mangel et al. [MMKH11] showed that reception can significantly degrade under certain conditions. As a comparison, 5G showed 90 % reliability at 1175 m and 875 m, respectively. Although a universally valid statement about DSRC and 5G range and reliability can barely be made, since they heavily dependent on a variety of different influence factors, it can still be concludes that 5G technology generally yields better performance.

Measurements concerning maximum throughput with IEEE 802.11p range from $2.7\,\frac{\text{Mbit}}{\text{s}}$ to $11\,\frac{\text{Mbit}}{\text{s}}$ per channel [CBW$^+$16, WDT$^+$13]. Average latency in typical CP use cases was measured by [RKD11] to range from 3 ms to 22 ms for small message sizes ($\sim 1.3\,\text{kB}$).

Despite limited performance of DSRC, current CP systems face another problem, which is related to **network utilization** with VANET topologies. Usually, clients in these networks establish complete pair-wise P2P connections among each other. Consequently, the number of connections is given as $N = \frac{n(n-1)}{2}$ according to *Metcalfe's law*, as opposed to $N = n$ when using a topology that involves a central server instance.

Due to these restrictions, recent publications tend to favor upcoming 5G technology [Bri19, 5G 16] over DSRC. For instance, [WL17] claims that *"development of V2X services in 5G makes much sense, and holds promises for the future"*.

A more detailed comparison between DSRC and 5G is presented in section 5.2.

4.2 Traffic Volume Estimation

This paragraph is to be seen as an excursion as its findings are a prerequisite for following sections. In order to formulate the scalability requirements for a cooperative perception system in section 4.3, a quantification of common traffic volumes is required. More precisely, this section aims to determine an average-case approximation of how many concurrent vehicles will usually be situated within a certain geographical area. Focus is placed on urban, inner-city scenarios.

4.2.1 Methodology & Results

An average-case value for the number of concurrent vehicles within $1\,\text{km}^2$ of urban area is to be determined. For its calculation, a heuristic two-step procedure is employed. Although based on many assumptions, this rough approximation is sufficient for this work's purpose as there is no need for a very precise quantity.

The city of Berlin is used as an example, as suitable open data is available for it. OpenStreetMap data files for Berlin[1] serve as a basis to compute road network length and

[1] `http://download.geofabrik.de/europe/germany/berlin.html`

average number of street lanes. They were fed into a PostgreSQL[2] database with the PostGIS[3] extension using *osm2pgsql*[4] and queried using SQL.

4.2.1.1 Assumptions

The following assumptions are made.

1. Average inner-city **driving speed** is $24\,\frac{\mathrm{km}}{\mathrm{h}}$ [For08]
2. Vehicles keep a **distance** of $s\ [m] = v\ [\frac{m}{sec}] * 1.8\ [sec] = v\ [\frac{km}{h}] * 0.5\ [h]$ [Wik19d]
3. An average **vehicle's length** is $4.2\,\mathrm{m}$
4. The **area** of Berlin Mitte is $21.25\,\mathrm{km}^2$ (see appendix section B.1)
5. The total **length of the road network** in Berlin Mitte is $159.13\,\mathrm{km}$ (see appendix section B.1)
6. The average number of **driving lanes** in Berlin Mitte is 2.4 (see appendix section B.1)

4.2.1.2 Step 1: Worst-case density at maximum utilization

First, a scenario is presumed in which the inner-city traffic network is at maximum utilization, that is, all driveable roads are completely occupied.
Given that every car drives the assumed average speed and keeps its minimum safety distance, the virtual length of every car is:

$$l_{virt} = 4.2\,\mathrm{m} + (24\,\frac{\mathrm{km}}{\mathrm{h}} * 3.6^{-1} * 1.8) = 4.2\,\mathrm{m} + 12\,\mathrm{m} = 16.2\,\mathrm{m} = 0.0162\,\mathrm{km}$$

In addition, the total length of driveable lanes is:

$$l_{road} = \frac{159.13\,\mathrm{km}}{21.25\,\mathrm{km}^2} * 2.4 = 17.97\,\frac{\mathrm{km}}{\mathrm{km}^2}$$

Accordingly, the maximum possible amount of concurrent cars within an area of 1 $\mathbf{km^2}$ is:

$$N_{max} = \frac{17.97\,\mathrm{km}}{0.0162\,\mathrm{km}} = \mathbf{1109\ [cars/km^2]}$$

[2]https://postgresql.org/

[3]http://postgis.net/

[4]https://github.com/openstreetmap/osm2pgsql

4.2.1.3 Step 2: Normalization with estimated average load factor

While the previous value of 1109 concurrent connected cars per km^2 is a worst-case assumption, Berlin's latest traffic census [Ver14] can be used as ground truth for estimating an average **load factor**. Using this load factor, the maximum, worst-case assumption can be scaled down to an estimated average value.

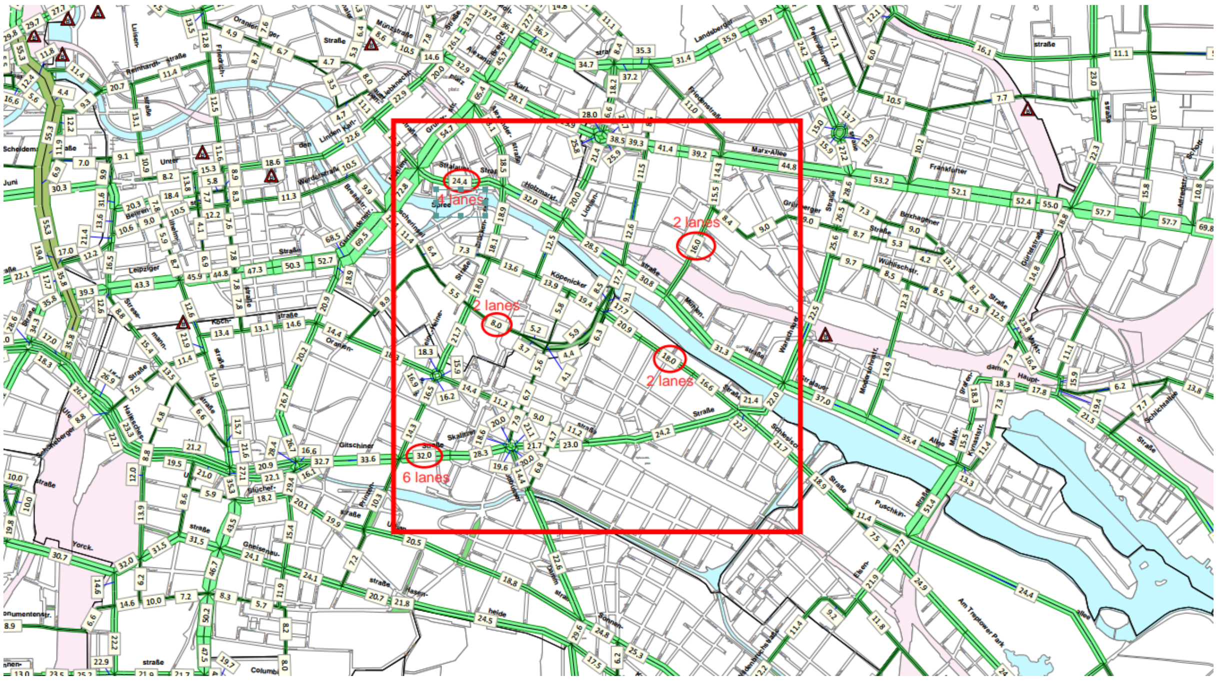


Figure 4.1: Traffic Census Map for Berlin Mitte [Ver14]

Figure 4.1 depicts an excerpt from the traffic census map, where the red rectangle marks the 21.25 km^2 area used in this evaluation. Red circles indicate five measuring points that were randomly selected for calculation of the average utilization factor. For each of these points a daily vehicle count is given as N_i in $\frac{vehicles}{24h}$, alongside the number of driving lanes m_i at that location.

$$N_1 = 32,000;\ m_1 = 6$$
$$N_2 = 18,000;\ m_2 = 2$$
$$N_3 = 16,000;\ m_3 = 2$$
$$N_4 = 8,000;\ m_4 = 2$$
$$N_5 = 24,000;\ m_5 = 4$$

Given the above assumptions for speed, vehicle length and safety distance, the time that one vehicle takes to pass a measurement point can be calculated:

$$\Delta t_{pass} = \frac{0.0162\,\text{km}}{24\,\frac{\text{km}}{\text{h}}} = 0.000\,675\,\text{h} = 2.43\,\text{s}$$

Accordingly, the maximum number of vehicles that can potentially pass a measuring point in 24 hours depend on the number of lanes at that measuring point and is given as:

$$N_{pot\_i} = \frac{24\,\text{h}}{0.000\,675\,\text{cars/h}} * m_i$$

For every point, its load- or utilization factor can be calculated as the fraction of actually measured cars and maximum potential cars:

$$\lambda_i = \frac{N_i}{N_{pot_i}}$$

For the above points it follows

$$N_{pot\_1} = 213,333;\ \lambda_1 = 15\%$$
$$N_{pot\_2} = 71,111;\ \lambda_2 = 25.3\%$$
$$N_{pot\_3} = 71,111;\ \lambda_3 = 22.5\%$$
$$N_{pot\_4} = 71,111;\ \lambda_4 = 11.25\%$$
$$N_{pot\_5} = 142,220;\ \lambda_5 = 16.9\%$$

and an average load factor of $\lambda_{avg} = 18.19\%$.
It can be concluded that, given the average load factor, an optimistic estimate for the average number of concurrent cars in Berlin Mitte might be given by:

$$N_{norm} = N_{max} * \lambda_{avg} = 1109\,\text{cars/km}^2 * 0.1819 = \mathbf{202\,cars/km^2}$$

.

4.2.2 Conclusion

Assuming that all cars were connected and participating in cooperative perception, $N_{norm} = \mathbf{202\,cars/km^2}$ is the amount of vehicles the system should, at minimum, be able to handle. For simplicity, this neglects the potential participation of pedestrians, cyclists, road infrastructure, etc. in CP. A more pessimistic value is given by $N_{max} = \mathbf{1109\,cars/km^2}$.

4.3 Goals & Requirements

Overall goal of this thesis is to propose, implement and evaluate a cooperative perception system, that facilitates the improvement of connected, autonomous vehicles' average perception quality. However, it is crucial that CP is always only used as extension for an already working, reliably and secure AD system.

The presence of cooperative perception must under no circumstances be required for the proper functioning of a self-driving car.

The presented architecture is supposed to overcome previous systems' limitations discussed in section 4.1 and be able to handle, at minimum, the average expected load determined in section 4.2. Therefor, functional (**F**) and non-functional (**NF**) requirements are defined for both parts of this work, namely environment modeling on the one hand and overall system architecture on the other.

4.3.1 Environment Modeling & State Representation

One milestone of this work is to propose a suitable meta model for traffic scenes alongside an appropriate representation format, both of which are expected to fulfill the following requirements.

F-M1: **Expressiveness.** The model should be expressive enough to capture all relevant aspects about a traffic scene, that are necessary to re-construct it with sufficient precision.

F-M2: **Openness.** The model should be open to include information on different levels of abstraction and allow for extension.

NF-M1: **Universality.** The model should be universal in such that it is independent of type (e.g. vehicle, pedestrian, RSUs, ...) and sensory of its observer and the involved fusion algorithms.

NF-M2: **Perspicuity.** The model should be globally valid and understandable, without requiring additional information about the observer, e.g. its local reference coordinate frame.

NF-M3: **Compactness.** The model should be efficient in terms of size.

4.3.2 Cooperative Perception System

The second major concern of this work is to propose an end-to-end solution for a cooperative perception system architecture and implementation, that fulfills the following requirements.

F-C1: **Holism.** The system should enable for end-to-end cooperative perception, including message exchange and sensor fusion, for different network patterns, including V2V, V2I and V2P.

NF-C1: **Range.** The system should enable for communication over at least 600 m in NLOS scenarios.

NF-C2: **Scalability.** The system should be able to handle at least 202 concurrent network participants per km^2 (see section 4.2) at an update frequency of 10 Hz (considered sufficient by [RKD11, TSG19])

NF-C3: **Efficiency.** The system should aim for low latency and low on-vehicle computation load.

NF-C4: **Reliability.** The system should avoid to have a single-point-of-failure.

4.4 Scope

As stated in section 4.3, it is within the scope of this thesis to design a comprehensive proposal, functioning implementation and realistic simulation of cooperative perception. However, the following is explicitly **out of scope**:

- The implementation does not have the extent, quality and resilience of a production-ready, well-tested system.
- While the proposed meta model allows for easy extension it is not complete in a sense that all potentially relevant classes, entities, attributes and relations are specified and implemented.
- Any aspects related to security, authentication, data integrity and extensive validation are disregarded.
- This thesis does conduct experiments or low-level investigations about 5G-, LTE- or DSRC network characteristics. Existing publications are used for reference instead.
- While a working proof-of-concept system is presented that fulfills the stated requirements (see section 4.3), elaborate (performance-related) optimizations of particular aspects (e.g. fusion, message generation, compression, ...) are out of scope.
- For the sake of simplicity, only two-dimensional environments are assumed. That is, vertically overlapping road scenes (e.g. with highway bridges) are not supported. However, the system could be extended to do so.

- Since detailed, realistic, 3D simulations are used for evaluation, no additional real-world experiments are conducted.

Chapter 5

Concept & Design

This chapter is thematically divided into four sections, for each of which a high-level concept or architecture is proposed, that aims to fulfill the requirements stated in section 4.3. Various design options are discussed and the proposed system's novelty in comparison to previous approaches is outlined conclusively.
Throughout this chapter, requirements from section 4.3 are referenced with the prefixes *F-* and *NF-*, respectively.

5.1 Environment Modeling & State Representation

As stated in section 4.3, a major goal of this thesis is to propose a common way to model and represent dynamic traffic scenes with the purpose of that model being used for cooperative perception. The following sections present challenges, requirements, design decisions and eventually a holistic concept.
In section 5.1.1, the decision for using high-level fusion is motivated and an overview of which information to be included in the proposed model is outlined. In section 5.1.2, modeling principles to be respected during the design phase are presented. Then, a basic structure / framework is proposed in section 5.1.3, before section 5.1.4 introduces probabilistic entity-relationship models as a way to uniformly incorporate rich semantics and a notion of uncertainty into the model. Eventually, section 5.1.5 unveils the final, comprehensive meta model to be used throughout the course of this work.

5.1.1 Object-Level Representation & Fusion

As explained in section 2.1.2, different levels of abstraction exist for sensor fusion. With regard to cooperative perception systems both low- and high-level fusion approaches are featured in literature. Chen et al. [CTYF19] favor the exchange of raw data over object-level information and argue that the latter requires a common reference object shared between two vehicles. As will be seen in later sections, this problem does not apply in the context of this work. Instead, high-level fusion is accompanied by benefits that heavily outweigh those of low-level fusion and is the only approach that allows to fulfill this work's requirements. Table 5.1 presents a detailed comparison of these two principles with regard to benefits and drawbacks. For each advantage the table states which requirement it helps to fulfill, respectively.

Advantages	**Disadvantages**
Significantly smaller data volumes, latency and network utilization (NF-M3)	Need for a common, shared model
Significantly lower computational load at observer vehicles (NF-C3)	Potential need for schema versioning
Independent of sensor type, characteristics and calibration (NF-M1, NF-M2)	
Support for different levels of abstraction (e.g. to include semantic information about pair-wise relations among traffic participants) (F-M1, F-M2)	
Allows for inference without further processing (NF-M2, NF-M3)	

Table 5.1: Advantages of High-Level Fusion over Low-Level Fusion for Cooperative Perception. Requirements are referenced, whose fulfillment the respective advantage enables for.

When using a high-level object model, the most essential part to be defined is the information the model is supposed to include. [PAKZ18] states that *"a traffic scene is described by the entities, their attributes and the relations among the entities"*. Following this definition and in order for the model to be as expressive (F-M1) as possible, the model is meant to include:

- State and topology of the immediate static environment and road
- State, static- and dynamic properties for both the ego vehicle itself and all surrounding traffic participants
- Relations among any kinds of entities within the ego vehicle's immediate surrounding (e.g. vehicle-vehicle-, or vehicle-traffic-light relations)

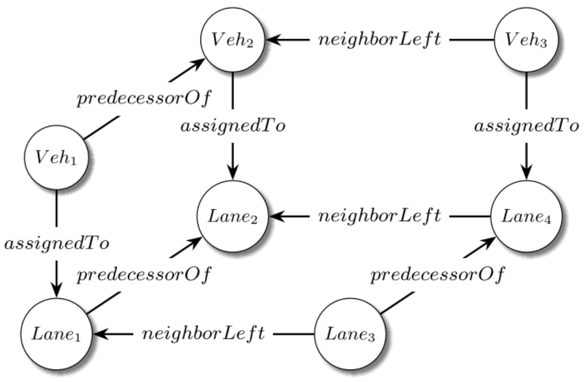


Figure 5.1: Illustration of a Graph of Semantic Relations between Traffic Participants [PAKZ18]

Especially the inclusion of semantic information – originally proposed by [KBSZ14] – is novel compared to all previously presented CP systems. Figure 5.1 depicts a minimal example of such relations.

5.1.2 Principles of Dynamic World Modeling

Modeling road traffic requires the ability to capture and integrate perceptual information of highly dynamic environments properly – a problem for which [CD93] states five essential principles.

P1: [Entities] in the world model should be expressed as a **set of properties**.

P2: Observation and model should be expressed in a **common coordinate system**.

P3: Observation and model should be expressed in a **common vocabulary**.

P4: Properties should include an **explicit representation of uncertainty**.

P5: Primitives should be accompanied by a **confidence factor**.

These principles are picked up again for the concrete model specification presented later and are referenced using their respective ***P*** prefixes throughout the course of following sections.
[CD93] also presents a "general framework for dynamic world modeling", depicted in fig. 5.2 (left). It illustrates the high-level process to transform heterogeneous types of observations into a unified model using a common vocabulary. The process includes a *Match-Update-Predict* cycle, the purpose of which is to enhance observations with evidence derived from previous observations and a prediction model. Although especially the *Match* step is quite essential for a real-world system, it is neglected in this thesis for the sake

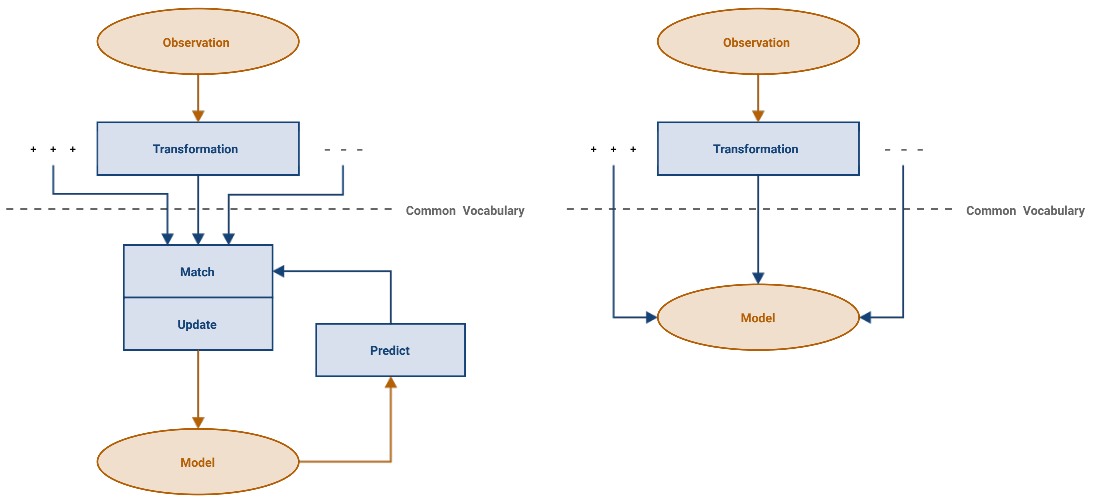


Figure 5.2: General Framework for Dynamic World Modeling [CD93] (left: original, right: simplified version used in this thesis)

of focusing on the higher-level system architecture. Instead, a simplified version of the process (right) is applied.

5.1.3 Discrete Environment Model with Occupancy Tiles

When attempting to model one's neighboring traffic environment, a mental distinction can be made into modeling the road network – including lane topology, traffic lights, sidewalks, etc. – and modeling static and dynamic obstacles, like other vehicles, pedestrians or trees. Both aspects are needed for a complete representation, as it is neither sufficient to only know the course of the road nor to only be aware of obstacles. Moreover, it is not sufficient to solely know an obstacle's position, but instead one is usually interested in more advanced properties, too. This work aims to combine all of these aspects in a shared model to enable them for being perceived cooperatively. Accordingly, the idea of [RKD11] to share an object list is combined with the concept of [LKC^{+}13] to share a discretized driveability map, also known as *occupancy grid*.

The very base of the proposed model is made up of cells of an **occupancy grid** with fixed dimensions. Such can be constructed by any observer (vehicle, RSU, etc.) and derived from any kind of perceptual sensor data. This simplification as a lowest common denominator facilitates universality (NF-M1). However, these basic information can be enriched with more complex features easily to support expressiveness (F-M1). This is achieved through the use of a graph-based meta model, as explained in section 5.1.4.
To work towards the perspicuity requirement (NF-M2) and to follow the principle of using a common coordinate system (**P2**), every cell is identified by a **QuadKey**. As a

consequence, referring to a cell's position in the world is independent from local, per-vehicle coordinates systems and from the GNSS / GPS coordinate frames alike. This prevents network participants from a multitude of expensive transformation operations.
Following this concept, the entire world map is recursively split into QuadTiles (see section 2.6) of a certain precision level (e.g. level 24, corresponding to square tiles of $\sim 2.39\,\mathrm{m}^2$). The precision level should be chosen in a way that it is fine-grained enough to distinguish between single pedestrians or even road signs. However, at the same time, it must not be too exact in order to save bandwidth and computation load.
Every cell of the occupancy grid – one of which is observed locally by every connected participant – then corresponds to a certain QuadTile. Besides its occupancy state, every tile can be augmented with information like the corresponding occupant actor, its relation to the overall road topology, etc.
Figure 5.3 illustrates the proposed *occupancy tile* concept. The blue grid is within the observation range of the turquoise vehicle and inherently part of larger tiles. Each cell's (= each tile's) state is determined through local sensor fusion involving different types of in-vehicle sensors and can be augmented with additional information. Eventually, that grid, entailing all relevant information, is shared with other CP participants.

5.1.4 Probabilistic Entity Relationship Model for Cooperative Perception

In the simplest case, each observed cell of the previously introduced occupancy grid holds only information on whether it is occupied or not. However, this small amount of information is usually insufficient to extensively describe a road scene. In addition, object-level static- (e.g. type, color and extent) and dynamic (e.g. velocity- and acceleration) information are desirable, e.g. about the respective occupying obstacle. This is in line with the requirements to support expressiveness (F-M1) and openness (F-M2) and can be taken even one step further. To do so, [KBSZ14, PAKZ18] motivate the idea to also include high-level semantic information about the relationships between different entities in a scene to reduce complexity and help generalization.

Up to this point, it is coarsely specified *what* to incorporate into the model: static and dynamic properties of each entity within the scene on different levels of abstraction – including semantics – using occupancy tiles as base- or root entities. The second step is to specify *how* to represent these. Previous requirements imply a structure in which **entities** have **attributes** (or properties, see principle **P1**) and **relations** of various types among different entities can exist. Additionally, in order to appropriately meet the inherently uncertain nature of a partially observable environment using imperfect sensory and to comply with principles **P4** and **P5**, some notion of **confidence** needs to be incorporated into the model. Some way is needed to probabilistically model an observer's belief in the

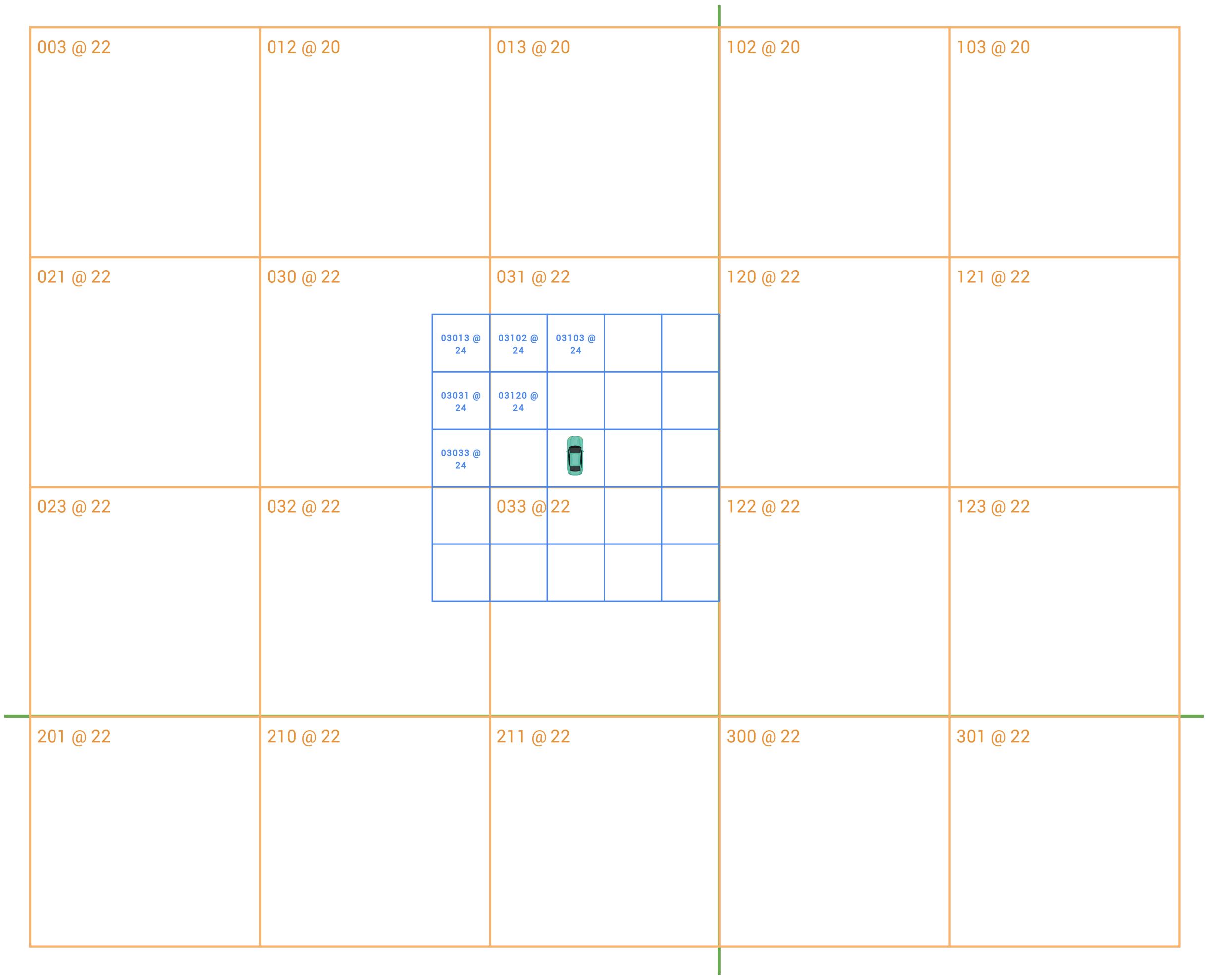


Figure 5.3: Illustration of an Occupancy Grid using QuadTiles

truthfulness of a particular property value or the existence of a certain relation between entities.

These needs lead to the introduction of **probabilistic entity relationship models** (PER models). Such are already used by [PAKZ18] for traffic scene representation and can be seen as an extension to regular entity relationship models (ER models). With ER models, an entity can either have an attribute or be relate to another entity. Both entity-attribute combinations and entity-entity relations can be represented as $\langle subject, predicate, object \rangle$ triples r with:

$$r \in \{\langle s,p,o\rangle | s \in \mathcal{E}, p \in (\mathcal{A} \cup \mathcal{R}), o \in (\mathcal{E} \cup \mathcal{L})\} \tag{5.1}$$

$\mathcal{E}$ is the set of all entities, $\mathcal{A}$ is the set of all attributes, $\mathcal{R}$ is the set of all entity-entity relations and $\mathcal{L}$ denotes a literal, i.e. a number, boolean value or string.

With PER models, as opposed to conventional ER models, every triple is now extended by an additional confidence parameter and thus becomes a **quadruple** r as part of the set of all potential quadruples $\mathcal{M}$:

$$r \in \mathcal{M} \text{ with} \tag{5.2}$$
$$\mathcal{M} := \{\langle s, p, o, \alpha\rangle | s \in \mathcal{E}, p \in (\mathcal{A} \cup \mathcal{R}), o \in (\mathcal{E} \cup \mathcal{L}), \alpha \in [0,1] \subset \mathbb{R}^+\}$$

With regard to this newly introduced confidence parameter α, [PAKZ18] distinguishes between *attribute-* and *structure uncertainty*, where the first describes sensor noise and the latter reflects uncertainty in the relational data. This is analogous to principles **P4** and **P5**, which suggest to model uncertainty for properties on the one hand and a confidence factor for entities on the other. However, this distinction is not considered useful in the context of this work and thus both probabilities are subsumed under a single notion of confidence.
An exemplary excerpt from an instantiated PER model, that could have been constructed based on observations of some ego vehicle, might look like this:

$$\langle \textit{obstacle\_25}, \textit{isOfType}, \text{“}\textit{small\_car}\text{“}, 0.921\rangle$$
$$\langle \textit{obstacle\_25}, \textit{hasVelocity}, (0.65, 0.42, 0), 0.448\rangle$$
$$\langle \textit{obstacle\_25}, \textit{isStoppedAt}, \textit{traffic\_light\_190}, 0.995\rangle$$

In the example, `obstacle_25` and `traffic_light_190` are entities, which were perceived, identified and tracked by an intelligent vehicle's sensory. `isOfType` and `hasVelocity` are attribute relations with attribute literals (string and three-dimensional numeric vector) and `isStoppedAt` stands for an entity-entity relation.
Such relations can be separated into seven classes: *inclusion*, *possession*, *attachment*, *attribution*, *antonym*, *synonym* and *case* [Sto93]. Three of those are considered relevant for modeling in AD contexts [PAKZ18]:

- **Inclusion:** Indicates structural- or spatial inclusion or class inheritance.
  Example: $\langle cross\_walk\_12, isPartOf, lane\_30, \alpha\rangle$
- **Attachment:** Indicates structural- or spatial connection or intersection.
  Example: $\langle car\_1250, drivesOn, lane\_30, \alpha\rangle$
- **Case:** Indicates interaction or dependence between entities or association of attributes and entities.
  Examples: $\{\langle car\_1250, isStoppedAt, traffic\_light\_190, \alpha\rangle, \langle bicycle\_370, hasColor,$ $\text{“}yellow\text{“}, \alpha\rangle\}$

While relations from any of these classes are represented equally in a PER model, the distinction can help to achieve a clean meta model or database design.

Using quadruples, a PER model is representable as a graph and thus can quite easily be searched, transformed and extended by inserting new relations. Also, the PER can – under certain conditions – be efficiently represented as a tensor [PAKZ18].

The graphical representation also allows for different types of inference. New facts about the local world could be inferred when used in combination with first-order logic. Also, situations might easily be compared in terms of similarity, which can be used for case-based reasoning. Nickel et al. [NMTG16] present a multitude of further techniques to perform relational machine-learning on knowledge graphs, which might prove promising in AD contexts as well and are worth to be considered in future work.

5.1.5 Final Model

The complete model proposed as part of this thesis is presented in fig. 5.5. It aims to be sufficiently comprehensive and includes all aspects needed for an in-depth evaluation of cooperative perception.

In the illustration, blue boxes denote entities, orange boxes stand for uncertain entity-entity relations and green circles are uncertain attributes (or entity-literal relations). Gray boxes are special in a way that they can not be subject to instantiation, but rather define the meta model's static structure as a class hierarchy. Filled blue boxes are not of a special meaning, except that they will usually occur particularly frequently.

In an instantiation of this model, quadruples include entities (blue) as subject or as subject and object and relations (orange) or attributes (green) as predicate.

It is worth noting that, in contrast to most prior work in this field, the occupancy state of a cell is modeled **ternary** instead of binary. An additional bit is used to distinguish between a cell being *free* or *occupied* or just *unknown*. This distinction adds valuable information for the participants of CP network, as an *unknown* (or *unseen*, *covered*, *beyond-LOS*) state can simply be substituted during the fusion step. In a situation where two vehicles are driving in line (depicted in fig. 5.4), the rear car might observe the cells occupied by the leader as actually occupied. However, there is no chance it can see through the leader car and thus all cells in front of it need to be communicated as being unknown, as opposed to either free or occupied.

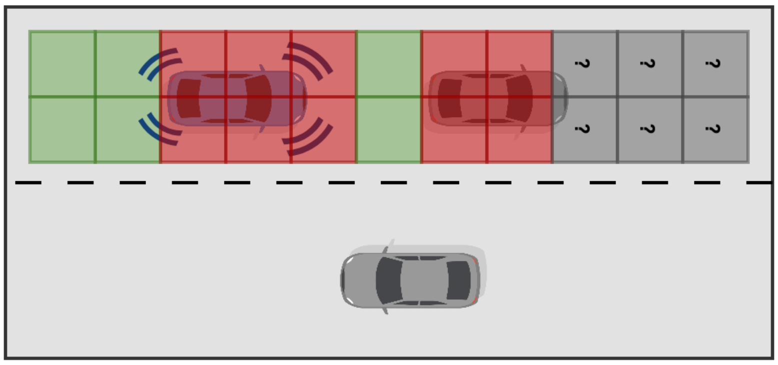


Figure 5.4: Blocked Line-of-Sight Scenario

In addition, the presented model is rich in a way that ...

- ... it defines a large variety of different entities and relations.
- ... it allows to exhaustively specify a cell's occupancy state.
- ... features different levels of abstraction (low-level attributes, high-level semantic relations).
- ... incorporates class hierarchies as known from object-oriented programming (e.g. $\langle Vehicle, isA, DynamicObstacle\rangle$).

A minimal instantiation of this model has to include an `OccupancyGrid` instance with an observer (`observedBy`) element (e.g. a `Vehicle`) and $1..n$ `GridCells`. How many cells a grid exactly consists of (grid size) can be configured and depends on its observer's sensor range. Each cell has to have at least its `state` and `positionHash` (as a QuadKey) defined. The ternary state can be represented using two bits. Its `positionHash` fits into 64 bits when using a QuadKey's integer representation (see section 2.6.1) and the confidence α would usually be implemented as a 32-bit float. Accordingly, one cell could be modeled compact (NF-M3) using 98 bits in an optimal, minimalist case.

5.1.6 Summary

In the previous sections a comprehensive, yet not complete model for dynamic traffic scenes was presented with a focus on cooperative perception use cases (see fig. 5.5). It enables to describe a scene by combining low-level attributes with high-level relational knowledge. It fulfills the requirements stated in section 4.3 and follows the modeling principles introduced in section 5.1.2.

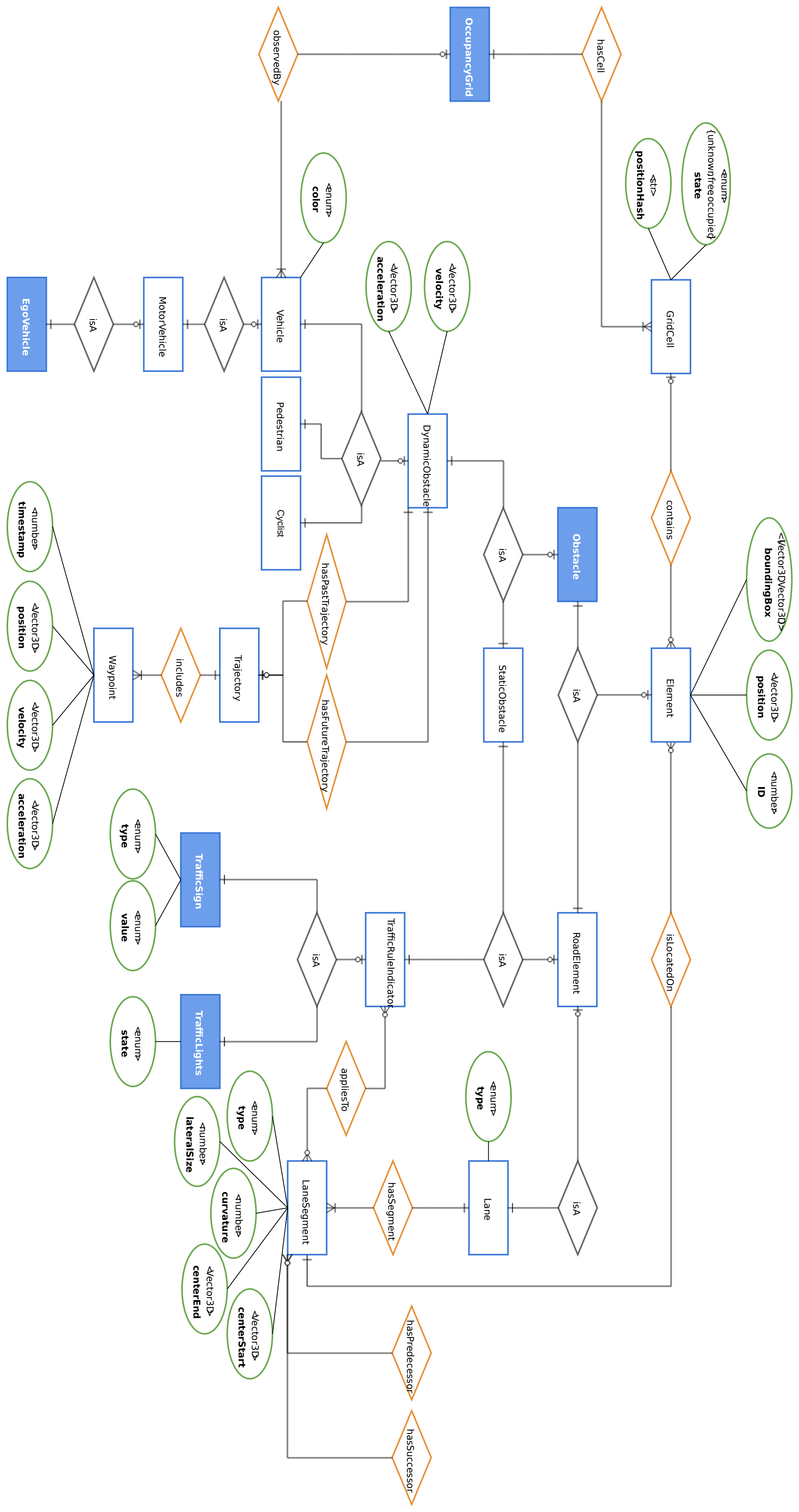


Figure 5.5: Comprehensive PER Model for Traffic Scenes

As a result, it is capable to uniformly and universally model road scenarios at a high degree of expressiveness, while embracing a notion of uncertainty or confidence and information on different abstraction levels. Key concepts used include **geo tiling** based on **QuadKeys** for spatially referencing cells of an **occupancy grid** and **probabilistic entity relationship models** as a structural framework.
However, future work in cooperation with expert groups is desirable to further extend the meta model.

5.2 Cellular Communication

Various downsides of DSRC-based ad-hoc networks for cooperative perception use cases were already discussed in section 4.1. They include issues arising from limited throughput and comparatively short range of IEEE 802.11p and similar technologies. Furthermore, P2P topologies usually imply a large communication overhead and result in high computational load for each participant. To overcome these limitations, an alternative approach is proposed that builds on 5G networks and a client-server topology.
This section briefly outlines key benefits and implications of 5G technology for V2X communication and motivates the decision for its use in the present CP system.

5.2.1 5G Usage Scenarios & Advantages

5G is a promising cellular network technology with the potential to overcome some of the limitation of DSRC discussed in section 4.1. Its characteristics, especially with regard to data throughput and latency, were introduced in section 2.5. In addition, [5G 16] presents a detailed, technical discussion of 5G vs. DSRC for V2X, while a higher-level comparison with respect to key implications for CP is presented in table 5.2.

Moreover, ETSI identified three different main usage scenarios for 5G [Eur], all of which require different key capabilities, as shown in fig. 5.6. The different usage scenarios are:

- **Enhanced Mobile Broadband (eMBB)** to be used for high-definition, low-latency multimedia content and mobile games on end-user devices and smartphones.
- **Massive Machine-type Communications (mMTC)** for the Internet of Things, which is characterized by low-power devices and low data rates.
- **Ultra-reliable and Low Latency Communications (URLLC)** for safety- and mission-critical applications, like autonomous driving.

Cooperative perception is a usage scenario that corresponds to the latter, as it especially requires low latency and high throughput. Accordingly, an appropriately capable 5G network infrastructure is essential for the proposed CP system to work.

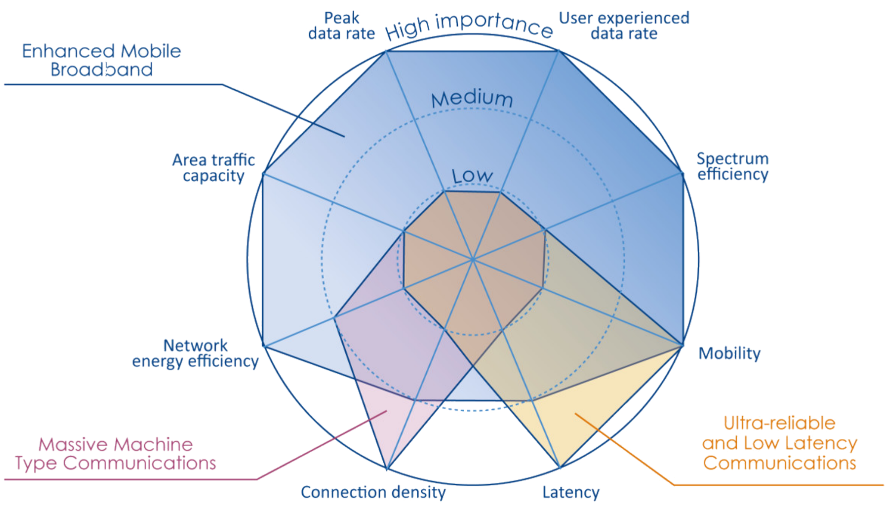


Figure 5.6: Key Capability Requirements for 5G in different Scenarios [Eur]

5.2.2 Vehicle-to-Network-to-Everything Communication Topology

Most prior work on V2X employs ad-hoc networks (VANETs) between participants, many of which imply a peer-to-peer network topology. That is, every participant connects to every other participant, following *Metcalfe's Law*. Downsides of such approaches were briefly outlined in section 4.1 and mainly refer to scalability and communication overhead.

In order to unburden participant vehicles (or pedestrian smartphones, RSUs, etc.), this work proposes the use of a **client-server architecture**. Every client connects to a central server instance, that is responsible for its current geographical area. It performs all computation tasks (mainly high-level sensor data fusion) and is responsible for collecting and broadcasting their messages. While the server node has to be very strong in terms of computational capacity, every client now only has to maintain one bi-directional connection and process (e.g. fuse) messages from one sender. Figure 5.7 illustrates both patterns in a scenario of $n = 4$ vehicles. The pattern of V2X now becomes, strictly speaking, V2N2X, as an intermediate network is involved. However, for the sake of simplicity, the term *V2X* will still be used to describe this new pattern.

Although 5GAA specify multiple transmission modes for 5G [5G 16] – including **direct communication** between network clients (V2V in the narrow sense of the word) – this thesis focuses on V2N-based communication to best counter the previously mentioned drawbacks.

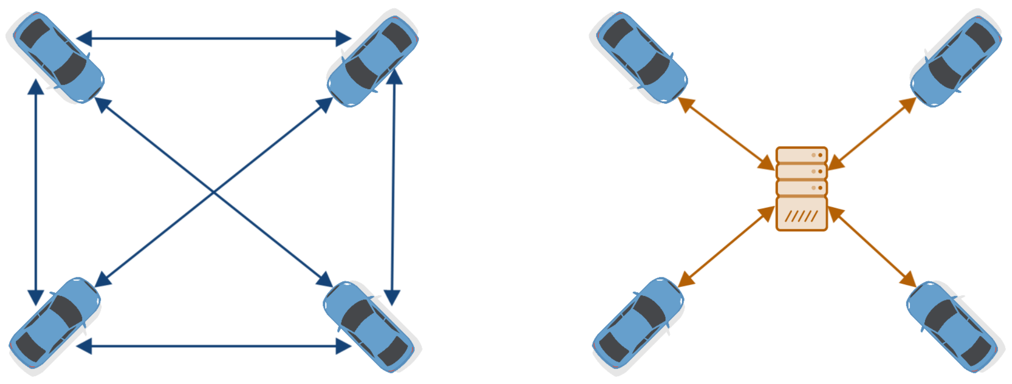


Figure 5.7: VANET vs. Client-Server Communication Topology

5.2.3 Summary

In contrast to most prior work, the CP system developed in the context of this thesis does not rely on DSRC-based VANETs, but on cellular 5G communication with centralized server instances instead. Motivation for this decision was given in previous sections and is additionally summarized in table 5.2.

VANETs + DSRC	**Client-Server + C-V2X**
+ Works anywhere	– Requires network infrastructure and -coverage
+ Free of charge	– Implies variable costs (e.g. data plan)
+ Network is solely dedicated for V2X	– Potentially shared network, e.g. with end-user smartphones
– Superlinear communication effort	+ Linear communication effort
– Linear computation effort for CP	+ Constant communication effort for CP
– 2.7-11 Mbps throughput [CBW+16, WDT+13]	+ > 65 Mbps throughput [Kav19, Wik19a]
+ 3-22 ms latency [RKD11]	– 10-20 ms latency (with Edge Node) [Wik19a, Qua18]
– ∼ 300 m NLOS range [5G 18]	+ ∼ 800 m NLOS range (direct communication) [5G 18] (→ NF-C1)
– P2P topology	+ P2P ("direct communication") or V2N topologies

Table 5.2: Comparison of DSRC-based VANETs and 5G-based Client-Server V2X Networks

In conclusion, 5G is a highly promising technology for cooperative perception and V2X communication in general. However, since it is a crucial requirement for the functioning of cellular V2X networks, their adoption will heavily depend on the market introduction of 5G. At the time of writing, 5G is still in its infancy. However, its expansion is being driven forward vigorously. For instance, German network operators are required to provide 5G coverage to 98 % of all households by 2022 [Deu19]. The authors of [CCS18], in turn, consider the market introduction of V2X applications an enabler for the establishment of 5G technology for mission-critical (URLCC, see section 5.2.1) usage scenarios and estimate them to take off from 2025 on. However, they put the usage of 4G Advanced, or 4.5G, as a fallback mechanism in perspective.

5.3 System Architecture

In section 5.1, a comprehensive environment model and a uniform representation format were developed before the use of 5G cellular networks was motivated in section 5.2. In this section, the overall software architecture and all of its components are elaborated.
While fundamental structures, patterns and interactions are specified, concrete technological choices are deferred to chapter 6, as the purpose of a software architecture is to *"facilitate development, deployment and operation in a way that leaves as many options open as possible, for as long as possible"* [Mar17b].

5.3.1 Central Fusion Nodes

As mentioned earlier, most existing cooperative perception solutions base on VANETs between their participants. Downsides of this concept with respect to both communication technology and network topology were presented in section 4.1. Only Ko-HAF [HKS$^+$19] constitutes an exception in that the proposed system utilizes cellular networks and involves a central server instance, the *Safety Server*.
Following their approach with the goal to overcome the limitations of P2P ad-hoc networks, this system's architecture will fundamentally base on a **central server component** at its core, also called fusion node in the following. Instead of P2P connections, the proposed system's topology follows a **client-server** communication pattern, potentially involving multiple servers. These will act as data brokers that are responsible for **(1) data collection**, **(2) fusion** and **(3) redistribution** (or broadcasting). Each of these steps is discussed in the following sections.
The approach to have multiple servers is essentially different from Ko-HAF and all other presented designs and results from the advanced requirements for computation performance due to higher frequency and accuracy. While the goal of Ko-HAF is to maintain

an always-up-to-date traffic map including static information and incidents, the amount of information exchanged per time is much higher with the present CP system, in which traffic situations are continuously being described and shared in high detail. In Ko-HAF, the amount of data transmitted per kilometer is 14-33 kB [HKS+19], while the data rate required for CP is potentially orders of magnitude higher (see chapter 7). Therefore, one central, global server instance is not sufficient to handle the system's high load. Also, one of the previously stated requirements is to avoid a single point of failure (NF-C4).

5.3.2 Geographical Partitioning

To meet the demand for high scalability (NF-C2) and reliability (NF-C4), including the avoidance of a single point-of-failure at system level, the concept of geographical partitioning, or **geo distribution**, is introduced. It is schematically depicted in fig. A.1 in appendix section A.3.1 and mainly enabled through the use of geo tiling (see section 2.6) and QuadKeys. While QuadKeys are already incorporated into the model to represent cells of an occupancy grid (see section 5.1), they also make up an essential part of the system architecture. Instead of having one global fusion node, e.g. per city, per district or per country, the concept is to have **one fusion node per tile**. A tile, in the sense of QuadKeys, could be of arbitrary size. For instance, a common setup could be to use level 16 tiles for geo distribution. That is, one fusion node would be deployed every $\sim$ $611\,\mathrm{m}^2$. All network participants within that area – and potentially those of all adjacent tiles – connect to this server instance and send and receive their data to and from it. When entering one of the adjacent tiles, the cars' connection would be updated to use the node corresponding to the new tile. Accordingly, a fusion node only has to handle a quite limited number of vehicles (pedestrians, etc.). With this technique, the density of deployed nodes might be adapted according to the expected average traffic density within that area. Usually, the number of fusion nodes per area will be higher in densely populated, urban areas than on the countryside. Section 4.2 presented a rough estimation for a typical, average amount of concurrent vehicles within an area and can be used as a guideline for deploying nodes. An in-depth evaluation of the fusion nodes' performance is conducted in chapter 7. Its results can help to choose an appropriate configuration for a system's λ_3 parameter (see below).

Overall, tiles of three different levels (λ) are used throughout the system.

- **Type 1** (typically $\lambda_1 \in [22, 24] \subset \mathbb{Z}$): Tiles used for cells of observed occupancy grids (blue cells in fig. 5.3).
- **Type 2** (typically $\lambda_2 \in [17, 20] \subset \mathbb{Z}$): Tiles used for a participant's range of interest, i.e. while a vehicle sends only its own occupancy grid, it receives data for a wider

range to extend its perception (orange cells in fig. 5.3).

- **Type 3** (typically $\lambda_3 \in [14, 16] \subset \mathbb{Z}$): Tiles used for geo distribution, i.e. one fusion node (or server, broker) per each of these tiles (green cell in fig. 5.3).

The aforementioned occupancy grids (set of all blue cells in fig. 5.3) conclude the concept of geographical partitioning. An occupancy grid does not correspond to tiles of a specific level, because its center position is continuous and its size depends on the observers' sensor range. The occupancy grid's size is larger than type 1 tiles (i.e. larger than its contained cells), but usually smaller than type 2 cells, because otherwise CP would be useless.

Example: A vehicle's perceptual sensors have a total range of 100 m and the occupancy cells (type 1 tiles) are configured to $\lambda_1 = 24$ ($\sim 2.4\,\mathrm{m}^2$). Accordingly, the grid size will be $\varnothing_{grid} = \left\lfloor \frac{100\,\mathrm{m}}{2.4\,\mathrm{m}} \right\rfloor = 41$ cells. Of course, the vehicle is interested in observations beyond its 100 m range, so it receives data from all its adjacent type 2 tiles, which are configured with $\lambda_2 = 19$ ($\sim 76.4\,\mathrm{m}^2$) in this example. Its range of sight has expanded to $\varnothing_{virt} = 3 * 76.4\,\mathrm{m} = 229.2\,\mathrm{m}$. The size of type 3 tiles is set to $\lambda_3 = 16$ ($\sim 611.5\,\mathrm{m}^2$), i.e. the vehicle will connect to a new fusion node every 612 meters. Each of these nodes is responsible for managing data for $4^{19-16} = 4^3 = 64$ type 2 tiles (or $4^{24-16} = 4^8 = 65536$ type 1 tiles).

5.3.3 Messaging & Further Considerations

Up to this point, a high-level system architecture was proposed and the concept of geographical partitioning was introduced as a solution to horizontal scalability. Complementing previous design decisions, this section formulates the exact communication pattern between clients (vehicles, pedestrians, etc.) and the fusion node from an application-layer perspective. In addition, further considerations regarding the messaging protocol are made.

In essence, two different patterns for communication in a client-server architecture can be thought of. First, there is the *request-response* (req/res) pattern, according to which a client explicitly “asks“ the server for certain information and potentially receives an “answer“ to its request. This type of communication is unidirectional and most used throughout today's web. The second pattern is *publish-subscribe* (pub/sub). Compared to req/res, it inverts the control flow to some extent. Instead of having the client (repeatedly) asking for new data, the server keeps providing such “automatically“ and just as it appears after the client had initially expressed its interest once. Depending on the concrete implementation, this type of communication can be uni- or bidirectional. Both patterns come with their respective advantages and downsides and are each suitable for certain use cases.

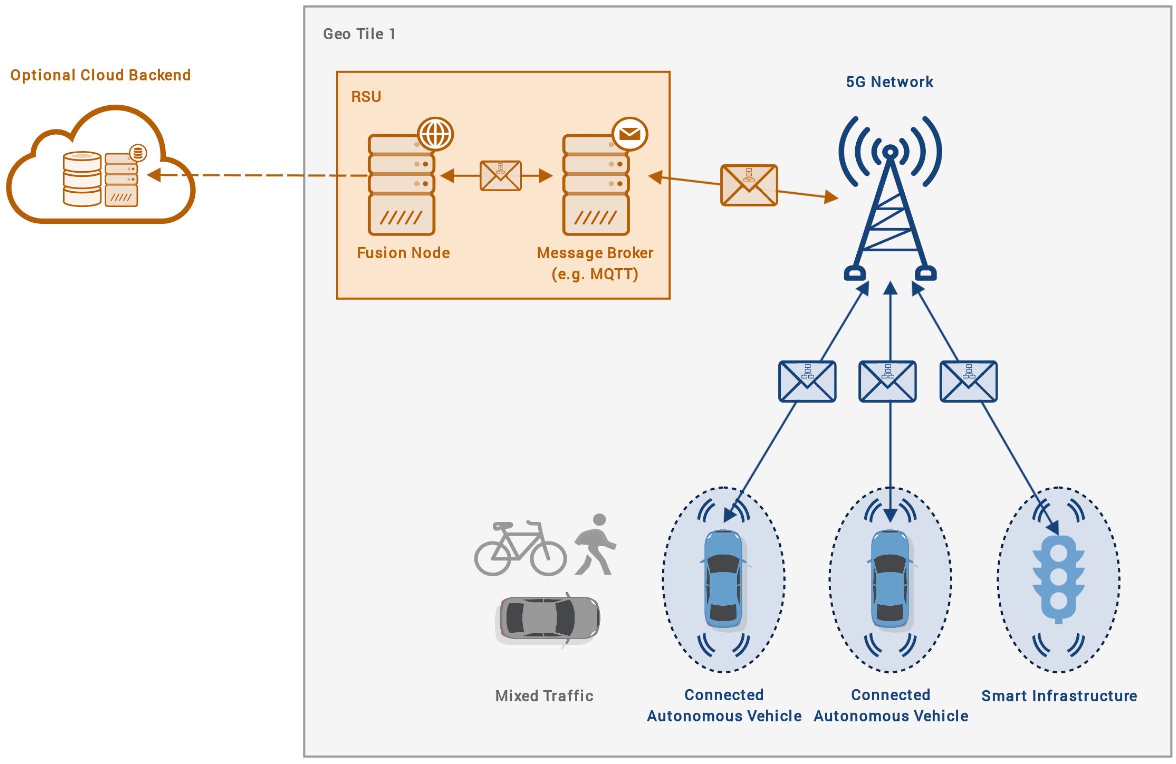


Figure 5.8: Software Architecture Schema

The system proposed in the context of Ko-HAF [HKS+19] utilizes both: req/res to send and receive static- and meta information and pub/sub to exchange time-critical data. For CP use cases, most information is time-critical. Assuming observations are continuously produced and fused at a fixed rate of 10 Hz for each observer, data needs to be distributed among the network as quickly as possible. Given these conditions, pub/sub shows to be the more suitable communication pattern, as it inherently supports these types of *data streaming* use cases and avoids additional overhead from repeatedly establishing new connections. As a consequence, communication between OBU and server is chosen to follow a **publish-subscribe** pattern with **periodic push** in the proposed system. The choice for a concrete pub/sub technology is deferred to chapter 6. However, since pub/sub systems are usually characterized by involving a separate **message broker**, such is already considered an integral system component. Figure 5.8 schematically illustrates the overall architecture for the proposed cooperative perception system.

Another important aspect to be agreed upon is the representation format of data on the wire, or, in this case, on a wireless connection. Although section 5.1.4 defined the environment model to be represented in form of a PER graph, no details have been provided on how to present that graph as a data structure. Since bandwidth is limited and latency must be kept low, a compact (NF-M3, NF-C3, cf. section 4.3) serialization

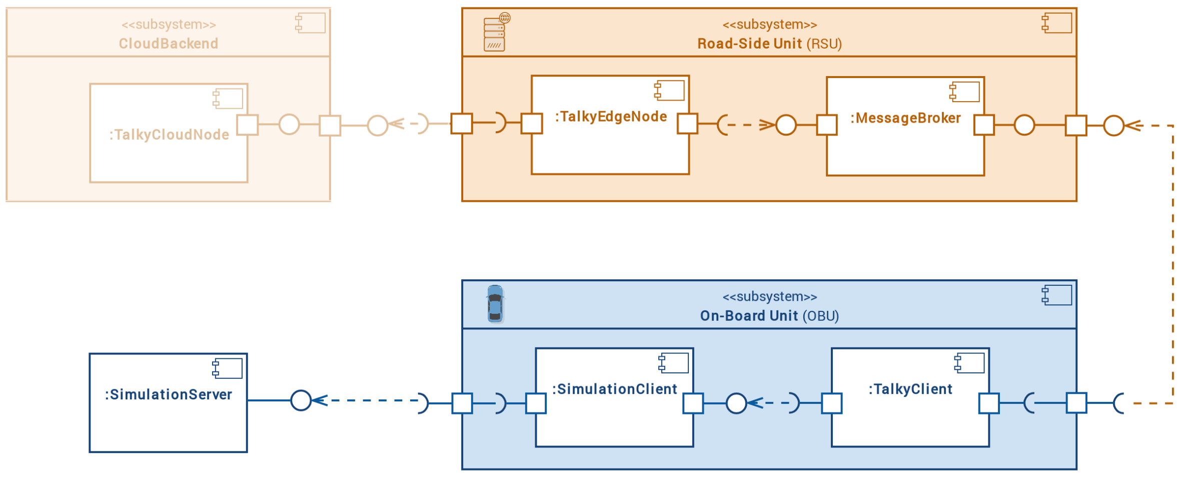


Figure 5.9: UML Component Diagram – System Architecture

format should be used. While the choice for a concrete technology is done in chapter 6, a **binary serialization format** might be favored over more verbose, text-based formats like XML or JSON.

5.3.4 Components Overview

After a high-level system architecture and respective communication patterns were elaborated, this section aims to provide an overview of all involved software components. Figure 5.9 schematically depicts respective relations among them. Each of these parts either already exists as third-party software or is implemented in the course of chapter 6.

C1: **Simulation Server.** An integral part of this work is to integrate, test and evaluate the proposed solution with a simulator. Although the choice for a particular simulator is deferred to chapter 6, it was early agreed on employing a photo-realistic 3D simulator with a programmable interface. All existing solutions follow a client-server model, i.e. the simulation is executed on a central software component which is controlled by one or more clients through an API. The environment, weather, physics, traffic, sensory and more are simulated on this central server and usually it is also responsible for 3D rendering of the scenes. Naturally, this component is only required for research purposes and would be omitted in a real-world deployment of the system.

C2: **Simulation Client.** This component lives within the **OBU subsystem**, i.e. is part of the collection of software that runs on-board of vehicles and other connected traffic participants. It belongs to the simulation infrastructure and is the counter-part of the previously mentioned simulation server, whose API it consumes. This client acts as an intermediary between simulation environment and the actual CP system and is responsible for receiving and converting sensor- and meta data and sending control commands. It might be implemented as a separate software component or integrated into the on-board software suite as a module and is supposed to be implemented in a way that it abstracts from an actual underlying perception system. In other words, it simulates a real in-vehicle perception system and therefore is also responsible for low-level sensor fusion. Ideally, the interface, including inputs and outputs, is so generic that the fact of it being part of a simulation is transparent from the outside. This component would be best realized following the *Bridge* and *Facade* design patterns (cf. [Eri13]).

C3: **Talky Client.** As a core part of the actual CP system on the vehicle side, this component fulfills a variety of tasks. First, it is responsible for reading locally fused sensor data from the simulation client and building an occupancy grid from it. After being augmented with additional meta information about the observer and context, the grid is eventually embedded into an instance of the previously mentioned PER model, then serialized and sent to the currently responsible message broker. Simultaneously, the Talky client listens for new incoming messages from that broker, i.e. new cooperative perception messages originating from surrounding network participants, that were aggregated and fused at the server. These messages are then deserialized into a PER model instance again to reconstruct the shared scene representation and then, once again, fused with the latest local observation. Details on how the fusion process exactly looks like are presented in section 5.4. In a real-world use case, the resulting fused scene representation would eventually be input to the planning module within the AD pipeline (cf. section 2.1.3) to help control the car more reliably. In this work, it is instead fed into the evaluation system directly to get insights about CP performance and more.

C4: **Message Broker.** The message broker is the first component to “sit“ on the server-side of the system, i.e. “on the other side“ of the intermediary 5G connection or, in other words, on the **RSU subsystem** (or server subsystem) in the above schema. It should be physically close to both the cell tower and the edge node (see below) to achieve low latency. In pub/sub systems a message broker is commonly employed in the middle between multiple applications that communicate among each other via messaging. In essence, it is responsible for maintaining layer 4 network connections with all connected devices and receiving and properly forwarding messages according to pre-defined policies. It abstracts from low-level, messaging-related tasks, including authentication and authorization, from business applications. Several implementations already exist, however, which one to use usually depends on which pub/sub protocol is chosen. Examples include, but are not limited to different AMQP[1]- and MQTT[2] brokers and are discussed in greater detail in chapter 6.

C5: **Talky Edge Node.** Also referred to as fusion node, this component implements the concept of an edge server in the sense of edge computing (cf. section 2.4) and is the second core part of the presented CP system besides the above mentioned Talky client. It is the central server instance that is deployed once per type 3 tile ()or *geo partition*). Its core responsibilities include to aggregate every connected participant's state representation (via the message broker) and fuse all of them to a single, combined state representation, which is eventually published back to all connected Talky clients again, following a *periodic push* pattern. As all PER models produced within the corresponding type 3 tile at a rate of 5-20 Hz are gathered here, this component has to be highly available, reliable, performant and equipped with a fast network connection.

C6: **Talky Cloud Node.** This component is part of a further level of abstraction on top of the previously presented edge node. Although it is optional and was not implemented as part of this thesis, it might be desirable in a real-world scenario. Similar to what the authors of [CM17] presented as third level of their multi-level cooperative perception scheme, the cloud node might be used for high-level tasks like navigation, traffic reporting or further analyses based on aggregated data from a multitude of different type 3 tiles. It could also be used as a central data lake for persistence. As the name suggests, this component would usually be deployed as one or few instances on the cloud and is another integral part of a typical edge computing architecture.

5.3.5 Summary

Unlike most previous takes on cooperative perception, the proposed system implements a client-server architecture with central fusion nodes to decrease network utilization and relieve the computational load on individual vehicles. To help scalability and reliability, the novel concept of geo partitioning using QuadKeys is introduced and in accordance with requirement NF-C4 (see section 4.3) the system also allows for redundancy, i.e. multiple fusion nodes per type 3 tile, to avoid a single point-of-failure at the partition level. Communication between network participants happens on a publish-subscribe basis with the central fusion node implementing periodic push.

[1]http://www.amqp.org

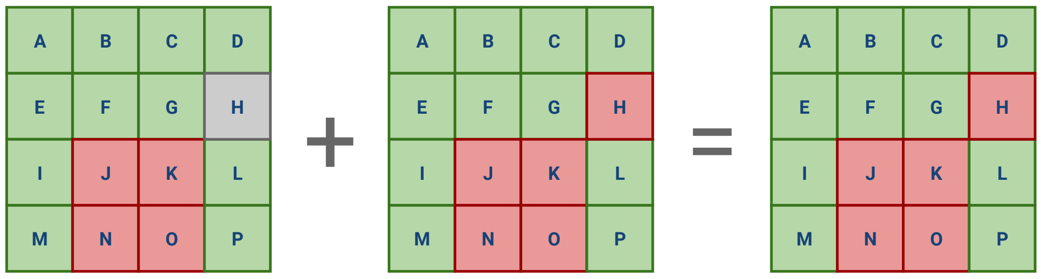
[2]http://mqtt.org

5.4 Fusion

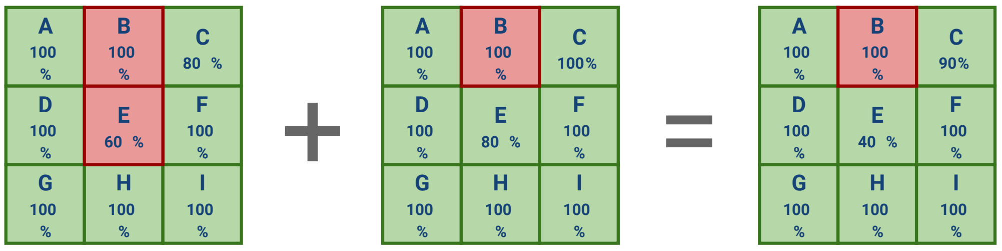
In addition to a model specification (section 5.1) and a communication- (section 5.2) and overall system architecture (section 5.3), a concept on how to perform high-level sensor data fusion is required. Although this part is not a central aspect in this work and might be elaborated in much greater detail and with more advanced methods in future research, basic considerations are still being discussed and an elementary concept is proposed. For the sake of clarity it is worth emphasizing that this section only concerns with the high-level fusion performed on client (C3) and edge node (C5), as opposed to low-level sensor fusion within the on-board perception module or, in this case, the simulation client (C1).

5.4.1 Goals

Essentially, the goals of fusion in a cooperative perception system are (1) to **supplement or impute missing data** and (2) to **increase the confidence** for existing data. The latter also includes to **resolve conflicts** and contradictory observations. (1) is essential for virtually perceiving a scene beyond the observer's own line of sight. (2) helps to increase accuracy and overcome sensor noise. Figure 5.10a and fig. 5.10b depict simplified examples, respectively, in which the occupancy state of cells are subject to fusion.



(a) Schematic Example for Supplementation through Fusion



(b) Schematic Example for Conflict Resolution through Fusion

In fig. 5.10a, vehicle X observes the occupancy grid on the left and vehicle Y the one on the right. Green cells are considered free, red cells are considered occupied and for gray

cells no state could be determined, e.g. due to sensor noise or because they were out of sight. Vehicle X failed to estimate the state for cell H, while vehicle Y measured it to be occupied. Accordingly, the fused grid includes cell H with a state of being occupied.
While confidences were omitted for the sake of simplicity in the first example, fig. 5.10b includes them in addition. In this scenario, attention is to be placed on cells C and E. While vehicle X is only 80 % confident about the state of cell C, vehicle Y is perfectly sure about it being free. Accordingly, the resulting state has an average confidence of 90 %. Moreover, a conflict exists for cell E. Vehicle X is 60 % confident, that it is occupied, while vehicle Y is 80 % confident of it being free under the open world assumption. When neglecting all other factors (like time), the average confidence for each cell's state might be used during the fusion to get a result of cell E being free with a confidence of 40 %. Please be aware that these examples are drastically simplified.

5.4.2 Problem Statement

Complementing the previous section, which informally stated the goals of fusion in the context of CP and provided two illustrative examples, a more formal problem definition is presented in the following.
As already mentioned in section 2.1.2, Elmenreich et al. define the problem of sensor fusion as *"the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually"* [Elm02].

More formally, for every type 3 tile with n participants (e.g. vehicles), let $M_i^{loc}(t), i \in [0..n]$ denote the model instance that was obtained by vehicle i at time t and assume the participant with $i = 0$ is the ego vehicle, whose perspective is taken in the following. As explained in section 5.1.4, the model M is essentially a set of quadruples r, which correspond to probabilistic relations of a graph. That is:

$$M_i^{loc}(t) := \{r_{i,0}(t) \;..\; r_{i,m}(t)\} \subseteq \mathcal{M}$$

$\mathcal{M}$ was defined in eq. (5.2) and denotes the set of all possible quadruples.
The goal of fusion is, given every participants local model M_i^{loc}, to get a global, augmented model M^{glob} that involves all other participants' relevant observations as well.

$$\begin{gathered}\text{Given } M_0^{loc}(t_0) \text{ and } \{M_i^{loc}(t) \mid i \in [1, n], t < t_0\}\\ \text{we look for a function } \varphi \text{ to compute } M^{glob}(t_0)\\ \text{s.t. } \psi(M^{glob}(t_0)) > \psi(M_0^{loc}(t_0))\end{gathered}$$

ψ denotes the evaluation or scoring function that compares the latent true environment state M^* to the observed one.
A more detailed description of how this augmented model is gained through fusion is given in section 5.4.6.

5.4.3 Scope

Before introducing the high-level fusion concept, the scope of fusion needs to be set. In principle, **all attributes and relations** in the model (see section 5.1.5) can be subject to fusion. To fuse attributes means to "merge" or aggregate multiple values for the same attribute, as demonstrated for the *state* attribute in fig. 5.10b. To fuse relations means to reason about their existence between two given entities. Naturally, this requires some kind of tracking and matching to uniquely identify entities across multiple observers, which is a non-trivial problem and subject to current research.
Attributes and relations might be categorized into classes, for each of which a separate fusion mechanism applies. For instance, when fusing cell occupancy states, a potential fusion technique might be to take a weighted average for the binary truthfulness of every possible state and determine the argmax subsequently. Similarly, when attempting to merge different values for an attribute that describes a spatial position, one might use the weighted average Euclidean distance during fusion. However, one might also come up with more advanced mechanisms. For the sake of simplicity, only the fusion of cell occupancy states is implemented in this work. However, the given framework can easily be extended to support more fusion subjects and mechanisms. The exact procedure of fusing cell states is presented in section 5.4.5.

5.4.4 Open- & Closed World Assumption

As explained in the previous section, all attributes and relations of the model can be fused in principle. However, different fusion mechanisms and functions need to be implemented for different types of attributes (e.g. numerical vs. categorical or temporal vs. spatial). Moreover, a major distinction has to be made for what to assume as an attribute's domain. Depending on the type or category of attribute, either the **Open World Assumption (OWA)** or the **Closed World Assumption (CWA)** is appropriate. The fusion result is fundamentally influenced depending on which one is used.
Taking a CWA means that – generally speaking – a logical statement, that is not known to be true, is considered false. Contrary to that, the OWA "allows" to have incomplete knowledge and does not consider *negation as failure* [Wik19c].
With respect to high-level fusion for CP, OWA vs. CWA plays an essential role when deciding what kind of fusion mechanism to use for different attribute classes. More pre-

cisely, it influences the way how confidences for entity-attribute or entity-entity relations are interpreted.
Generally, one could distinguish between attributes, whose domain is known and such, whose domain is not known. For the former, one could apply either CWA or OWA, while for the latter, the OWA has to be taken as a basis. For instance, while the domain for a ternary occupancy state is completely known (*free*, *occupied* and *unknown*), the number of possible "attribute" values (i.e. entities as objects in a quadruple) is infinite for an entity-entity relation between different road participants. During the fusion, it is unknown how many potential vehicles, pedestrians and other traffic participants are in a scene and could potentially occupy a certain cell or not, so only the OWA is appropriate.

5.4.4.1 Example

Assume the observations from two different observers for the same situation are present and their estimations of the occupancy state of a certain cell shall be fused. According to the previous model definition, both observers report exactly one quadruple for this cell's state each. Observer A reports $r_{1,A} = \langle cell_i, hasState, free, 0.9\rangle$ and observer B reports $r_{1,B} = \langle cell_i, hasState, occupied, 0.8\rangle$. Taking the CWA, the reported confidences may be interpreted as probabilities and since all possible states are known, the trivial assumption could be made that the two remaining unknown state values equally share the remaining probability mass. As a consequence, the following quadruples could be inferred:

$$\begin{aligned} r_{2,A} &= \langle cell_i, hasState, occupied, 0.05\rangle \\ r_{3,A} &= \langle cell_i, hasState, unknown, 0.05\rangle \\ r_{2,B} &= \langle cell_i, hasState, free, 0.1\rangle \\ r_{3,B} &= \langle cell_i, hasState, unknown, 0.1\rangle \end{aligned}$$

Applying a simple arithmetic average over the confidences of all observations for each possible state yields:

$$\begin{gathered} r_{1,\text{fused}} = \langle cell_i, hasState, free, \alpha_1\rangle \\ r_{2,\text{fused}} = \langle cell_i, hasState, occupied, \alpha_2\rangle \\ r_{3,\text{fused}} = \langle cell_i, hasState, unknown, \alpha_3\rangle \\ \text{with } \alpha_1 = 0.5, \alpha_2 = 0.425, \alpha_3 = 0.075 \end{gathered}$$

Taking the OWA, confidences can not be interpreted as probabilities, as the potential number of values is infinite. Instead, the trivial fusion approach from above would yield these confidences when assuming an open world:

$$\alpha_1 = 0.45, \alpha_2 = 0.4, \alpha_3 = 0$$

As mentioned earlier, only the fusion of occupancy states is implemented in this thesis, for which the open-world assumption is used consistently.

5.4.5 Mechanism: Time-Decayed Weighted Average

This section describes how multiple observations for a single discrete, categorical attribute are fused. For convenience, continuous attributes are disregarded and the focus is placed solely on the fusion of cells' occupancy states. This attribute's domain is discrete and finite (ternary). For instance, the statement that a cells is *free* is either true or false and an additional notion of uncertainty is incorporated by passing along a confidence value. The confidence value represents either attribute- or structure uncertainty (cf. section 5.1.4). Despite this confidence, a fusion function also needs to respect the observation's temporal delay, i.e. its "age". In a cooperative perception system, the presence of a transmission lag is inevitable and mainly arises from network latency. Accordingly, an observation is always outdated already when being fused. This degree of "outdatedness" is referred to as "reliability of perception" by [LKC+13], whose authors suggest to exponentially decrease it with increasing transmission lag. The description of a scene that is a few milliseconds old might still be quite accurate, but as it gets older – potentially several seconds – the scene has most likely changed too much to still rely on that state representation.
The fusion function φ is defined as a mapping from a set of model instances drawn from $\mathcal{M}$ (cf. section 5.1.4) – i.e. collections of semantic quadruples from n observers over multiple time steps – and corresponding discrete temporal delays $\Delta t = t_{now} - t_{obs}$ to a new model instance.

$$\varphi : \mathcal{M}^n \times \mathbb{N}^n \rightarrow \mathcal{M} \tag{5.3}$$

For illustration purposes, let $S(t)$ be an aggregated set of tuples of model instances and corresponding temporal delays at time t from all present observers and all relevant past time steps.

$$S(t) = \bigcup_{i=0}^{n} \bigcup_{u=t-\delta t^{max}}^{t} \langle M_i^{loc}(u), t - t_i \rangle \tag{5.4}$$

n denotes the number of observers and δt^{max} is a user-defined parameter that specifies the maximum age of observations to include during fusion. Consequently, the set $S(t)$ consists of all present observations (= PER model instances) from all present observers within the last $t - \delta t^{max}$ time steps. A fused model instance at time t is then given as:

$$M^{glob}(t) = \varphi(S(t)) \tag{5.5}$$

Evaluating φ involves to evaluate the respective “sub-“functions responsible for fusing the values of all certain entity-attribute ($\langle e, a\rangle$) or entity-entity ($\langle e1, e2\rangle$) combinations. Moreover, the following function is defined to determine a **temporal decay factor** for every observation to weigh it by its degree of “outdatedness“, just as proposed above.

$$decay(t_i) = e^{-\lambda * \Delta t} = e^{-\lambda * (t_{now} - t_i)} \qquad (5.6)$$
$$\text{with } \lambda > 0$$

In this thesis, the above function is used globally throughout the entire fusion procedure to incorporate transmission lag.

For a $\langle e, a\rangle$-combination whose value domain is categorical and finite, the **weighted arithmetic mean** can be used to aggregate (i.e. fuse) multiple evidences or observations. This is the method of choice implemented in this work to fuse occupancy states. Further fusion mechanisms for other types of $\langle e, a\rangle$- or $\langle e1, e2\rangle$ combinations are considered out of scope. In search to fill the placeholders θ_1 an θ_2 of a $\langle e, a, \theta_1, \theta_2\rangle$-tuple, one could follow a two-step procedure to determine the aggregated confidence value and actual attribute value.

$$maxconf_{e,a} = \max_{v \in dom(\langle e,a\rangle)} \sum_{j=0}^{m} OWA(j, v) * decay(t_j)$$
$$maxval_{e,a} = \arg\max_{v \in dom(\langle e,a\rangle)} \sum_{j=0}^{m} OWA(j, v) * decay(t_j)$$

In the above equation, the function OWA is defined as

$$OWA(i, v) = \begin{cases} \alpha_i & v \in r_i \\ 0 & else \end{cases}$$

and responsible for returning either the i-th quadruple’s confidence, if the quadruple’s attribute value (e.g. *free* in the case of a *state* attribute) equals v or zero otherwise. Moreover, m is the number of observations to be included in the current fusion round. Sticking to the previous notation, α_i is the confidence value (fourth component) of a relation quadruple, which again in part of a model instance M.
The fused quadruple for this specific $\langle e, a\rangle$-combination would be:

$$r_{i,fused} = \langle e, a, maxval_{e,a}, maxconf_{e,a}\rangle$$

5.4.6 Architecture: Doubly Updated Merging

While the previous section gave a thorough description of how a single entity-attribute combination might be fused, this section returns back to a higher-level perspective and examines the fusion process in its whole again.

The concept of *doubly updated merging* is introduced as a technique to address the fact that an observation is delayed twice on its way through the CP system. First, as a local observation $M_i^{loc}(t)$, it experiences a transmission lag Δt_1 while being transferred from the on-board Talky client via the message broker to the fusion- / or edge node. After it was fused at the server and became part of $M^{glob}(t+\Delta t_1+\epsilon_1)$ it is transmitted all the way back to the client, which causes another delay Δt_2. While the server considered Δt_1 as a decay factor during fusion, the client needs to incorporate Δt_2 in addition. Therefore, the client performs another round of fusion, in which $M^{glob}(t+\Delta t_1+\epsilon_1)$ and its latest local $M_i^{loc}(t+\Delta t_1+\epsilon_1+\Delta t_2+\epsilon_2)$ are merged into the final M_i^{final}. This process is depicted as a sequential schema in fig. 5.11.

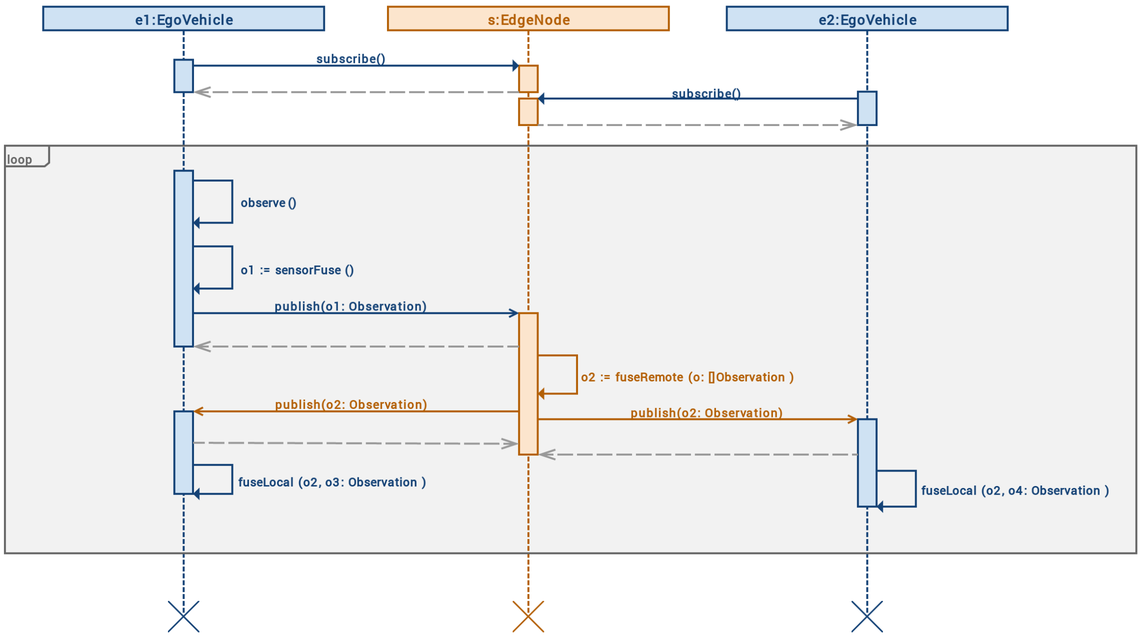


Figure 5.11: UML Sequence Diagram – High-Level Fusion

Given the definition of relation quadruples r in eq. (5.2) and the definition of set $S(t)$ in eq. (5.4), all three different state representations (or model instances) produced in the course of one iteration of *doubly updated merging* can be formally stated as follows.

$$\textbf{Local:} \quad M_i^{loc}(t_0) = \{r_{i,0}(t_0) \;...\; r_{i,m}(t_0)\} \subset \mathcal{M} \tag{5.7}$$

$$\textbf{Globally Fused:} \quad M^{glob}(t_1) = \varphi(S(t_1)) \tag{5.8}$$

$$\textbf{Locally Double-Fused:} \quad M_i^{final}(t_2) = \varphi(M^{glob}(t_1) \cup \{M_i^{loc}(t_2)\}) \tag{5.9}$$

$$\text{with } \delta t^{max} > 0, t_2 > t_1 > t_0$$

5.4.7 Summary

Previous sections presented a very basic concept of how to perform high-level sensor fusion in a cooperative perception system, which is later implemented and evaluated in chapter 6 and chapter 7.
However, since fusion is not the core aspect of this thesis, the previously presented techniques are only rudimentary and could be advanced significantly. Current limitations include that fusion was only defined for one type of properties. Also, the above fusion approach assumes that every time lag is precisely known, i.e. all involved devices' clocks are perfectly synchronized and additional delays (e.g. from sensor to controller) are disregarded. Elaborate methods to account for unknown time already exist in literature [JU05] and could be applied as part of future work. Moreover, tracking and matching of entities across multiple frames and multiple observers is not considered at all. Finally, while the taken approach to simply decrease the influence of temporally "old" observations is the simplest, better performance might be achieved through imputation of missing data or extrapolation of past measurements to current time [CTYF19].

5.5 Conclusion

Previous sections conceptualized the end-to-end design of a cooperative perception system with regard to all relevant aspects. First, a modeling approach and an accompanying representation format for dynamic traffic scenes was proposed. Subsequently, the use of cellular communication and a client-server network topology was motivated before a comprehensive distributed software architecture was proposed. Eventually, an elementary concept for high-level sensor fusion to be integrated with the new model and system architecture was presented. The proposed system design addresses all requirements stated in section 4.3. Especially, it is holistic (F-C1) in such that covered aspects range from message representation over communication, scalability and fault-tolerance throughout to high-level sensor fusion. Chapter 6 explains how all of these concepts were implemented in software.

Chapter 6

Implementation

The previous chapter addressed different aspects of a cooperative perception system conceptually. First, a uniform, expressive and extensible way to model traffic scenes for cooperative perception was proposed alongside an appropriate representation format for it. Different characteristics and benefits of 5G cellular networks were discussed before a high-level system architecture, involving a multitude of different modular software components, was elaborated. Eventually, a concept was presented on how to combine observations from different actors, while taking temporal delay and uncertainty into account.

The purpose of this chapter is to pick up the previously presented concepts and techniques and explain how they were implemented in software. This includes a discussion about various technological choices, the use of appropriate software design patterns and fundamental performance-related considerations. Figure 6.1 shows the component diagram from section 5.3.4 again, but now includes implementation-specific technologies in addition. It serves as a guideline for this chapter, as the implementation of each individual components is explained subsequently.

During the implementation process, design- and component principles presented in [Mar17b] were followed with the purpose to develop clean, modular and maintainable software components.

6.1 Meta Model, Representation- & Message Format

6.1.1 Object-Oriented Model

The meta model presented in section 5.1 is a probabilistic entity relationship model, i.e. essentially an ER model that is extended to incorporate a notion of structural- and relational uncertainty. As explained earlier, it can be viewed as a graph in which entities relate to

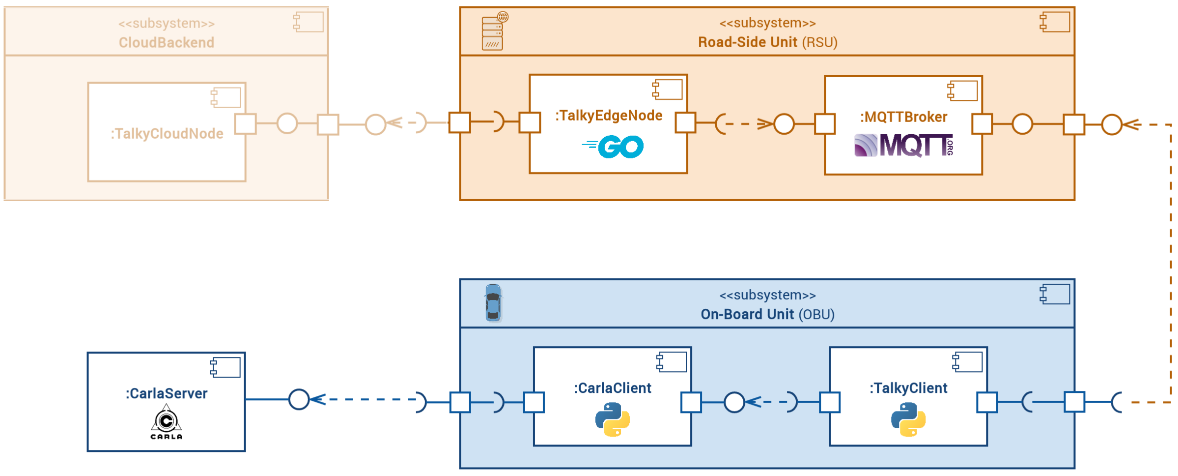


Figure 6.1: UML Component Diagram – Implementation

other entities or attributes with a certain probability or confidence. Accordingly, the graph can also be represented as a collection of quadruples $r_i = \langle subject, predicate, object, \alpha \rangle$ with α being the corresponding confidence factor.

In code, this graphical structure is implemented as a composition of pure object-oriented classes. As Python was chosen as the primary programming language, the built-in Python class system is employed. Each type of entity corresponds to a certain class, which inherits from an abstract `Entity` class. Relations (both entity-entity and entity-attribute) are implemented as sub-classes of an abstract `Relation` class, which is attached to its object as a class member and holds members for its object (either a literal attribute or another entity) and the corresponding confidence. Deviating from the quadruple structure, an exception is introduced for convenience to additionally allow for the specification of attributes without uncertainty as well. Such are simply implemented as ordinary class members and should only be used to encode meta data, but not actual observations. Listing 1 depicts a simplified example of this modeling schema.

Most of the entities, attributes and relations from the final model, presented in section 5.1.5, were implemented according to the above class schema and encapsulated as a re-usable Python library.

```
class PEREntity(ABC):
    pass

class PERRelation(ABC):
    def __init__(self):
        self.obj: Union[PEREntity, Any] = None
        self.confidence: float = 0.0

class OccupancyCell(PEREntity):
    def __init__(self):
        self.hash: int = 0
        self.state: Union[PERRelation[OccupancyState], None] = None

class OccupancyState(Enum):
    FREE = 1
    OCCUPIED = 2
    UNKNOWN = 3
```

Listing 1: Example Implementation of PER Entities and Relations

One might wonder why a cell's `hash` attribute is declared as an integer data type, even though QuadKeys (cf. section 2.6.1) are strings. This is due to the fact that the implementation takes advantage of an optimization in representing QuadKeys. In addition to using ASCII encoding, where every character is one byte, QuadKeys can also be represented as integers for better space efficiency. In integer representation, the length of a key is only limited by the underlying data type (usually 64 bits). The size complexity is then given as $O(1)$ compared to $O(n)$ with strings (n being the precision level, or key length).

6.1.2 Serialization Format

The object-oriented schema presented in the previous section is used as a module throughout all related Python code. However, instances of these classes only exist within the scope of a certain process. In order to transfer these information across different programs, which are potentially even written in different programming languages, a language-agnostic, commonly understandable format is needed. In technical terms, objects must be *serialized*, then transmitted and *deserialized* again afterwards. Multiple common serialization formats exist, which differ in certain properties. Some are text-based and potentially also human-readable, others are binary formats and only understandable by machines. While text-based formats are generally more common and easier to work with, section 5.3.3 motivated the use of binary formats for performance- and efficiency-critical applications like CP.

Common binary serialization formats include Apache Thrift[1], Cap'n'Proto[2], Flatbuffers[3] and Protocol Buffers[4] (Protobuf). Most of them follow the principle of first defining a static class (or message) schema, which is then compiled to language-specific code to be used by different applications equally. During serialization, objects and attributes are condensed to an efficient binary representation, which can usually be accessed as a byte stream or -array subsequently. Since all of these formats are relatively similar, a detailed comparison and evaluation is out of scope.

In a first iteration **Cap'n'Proto** was used. Reasons for this decision included the high serialization performance claimed on the authors' website and the framework's novelty. However, as this work progressed, it became clear that serialization was a major performance bottleneck. In search of alternatives to Cap'n'Proto a brief evaluation[5] revealed superior performance of **Protocol Buffers** format in comparison. In a simplified example, Protobuf was able to serialize 7490 messages/s on average, compared to 4129 messages/s with Cap'n'Proto (see section C.1). Moreover, the average message size with Protobuf appeared to be up to ~ 47 % smaller for the same payload. The benchmarking was done using the Go programming language, Google's reference implementation of Protobuf for Go and the most common third-party open-source Go implementation of Cap'n'Proto[6]. The performance improvement of ~ 25 % and potential size decrease of up to 47 % led to the decision to refactor existing code and use Protocol Buffers for serialization over the course of this work.

Listing 2 gives an example for a simplified message schema definition in Protobuf and Cap'n'Proto, respectively.

6.2 Simulation Environment

Since the integration and evaluation with an autonomous driving simulator is one of the core goals of this thesis, an appropriate simulator must be chosen. A commonly used option is **SUMO**[7] (Simulation of Urban Mobility). However, it focuses on traffic simulation and is not particularly built for in-depth simulations of autonomous driving. Instead, a variety of high-detail, photorealistic 3D simulators have established recently. The most commonly used options are the following.

[1]`https://thrift.apache.org/`

[2]`https://capnproto.org/`

[3]`https://google.github.io/flatbuffers/`

[4]`https://developers.google.com/protocol-buffers/`

[5]`https://github.com/n1try/talkycars-thesis/tree/master/src/evaluation/serialization`

[6]`https://github.com/capnproto/go-capnproto2`

[7]`https://sumo.dlr.de/`

```
syntax = "proto3";

enum OccupancyState {
    FREE = 0;
    OCCUPIED = 1;
    UNKNOWN = 2;
}

message OccupancyStateRelation {
    float confidence = 1;
    OccupancyState object = 2;
}

message OccupancyCell {
    uint64 hash = 1;
    OccupancyStateRelation state = 2;
}

message OccupancyGrid {
    repeated OccupancyCell cells = 1;
}
```

```
@0xc77abe9e219ad98d;

enum OccupancyState {
    free @0;
    occupied @1;
    unknown @2;
}

struct OccupancyStateRelation {
    confidence @0 :Float32;
    object @1 :GridCellState;
}

struct OccupancyCell {
    hash @0 :UInt64;
    state @1 :OccupancyStateRelation;
}

struct OccupancyGrid {
    cells @0 :List(OccupancyCell);
}
```

Listing 2: Exemplary schema definitions in Protocol Buffers (left) and Cap'n'Proto (right)

- **AirSim** by Microsoft [SDLK17] is an open-source (~ 9,300 GitHub stars, MIT license) 3D simulator for autonomous cars and drones based on the Unreal 4 game engine[8]. It has inherent support for Reinforcement Learning, allows to connect various kinds of external control devices and has a rich C++ and Python API to program it. Supported sensors are RGB camera, IMU, GPS, magnetometer, barometer, a custom distance sensor and LiDAR.

- **Carla** [DRC$^+$17] is an independent open-source project (3,600 GitHub stars, MIT license). The 3D simulator is based on Unreal 4, has C++ and Python APIs and features a multitude of sensors, including RGB camera, depth camera, IMU, GPS and LiDAR. In addition, is has multi-agent support, i.e. allows multiple Python- or C++ clients to connect to the simulation server and also offers an integration with the Autoware AV stack[9].

- **Carmaker**[10] by IPG Automotive GmbH is a proprietary 3D simulator featuring various integrations with hardware platforms and tools and offers an proprietary

[8]https://www.unrealengine.com

[9]https://www.autoware.ai/

[10]https://ipg-automotive.com/de/produkte-services/simulation-software/carmaker/

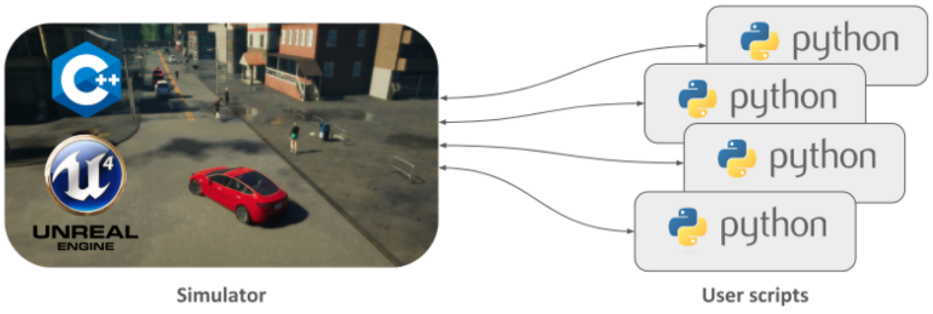


Figure 6.2: Client-Server Schema in CARLA [Car]

and a MATLAB programming interface.

- **LGSVL**[11] by LG is the latest of these 3D simulator projects, as development started in 2019. It is open-source ($\sim$ 700 GitHub stars, custom license), written in C# and based on the Unity game engine[12], offers integrations with Autoware and Baidu's Apollo framework[13] and a Python API. Supported sensors include RGB camera, IMU, GPS, LiDAR, radar and CAN bus. While this project appears to be the most ambitious and promising one, it is still in quite early development and therefore partially unstable.

Mainly because of its extraordinarily rich and intuitive API, the large number of simulated sensors, the inherent multi-agent support an a vibrant open-source community Carla was chosen to be used as a simulation environment in this work. In addition to the features mentioned above, it also includes seven pre-defined high-detail maps, each of which represents a different scenario (e.g. urban or rural environments). Maps can be exported in the OpenDrive format and thanks to an integration with the RoadRunner[14] software suite, custom maps can be created and imported easily. In addition to vehicles pedestrians and cyclists are supported as traffic participants as well and the open-source community has provided automated controllers and navigation scripts for each of these actor types. Figure 6.2 schematically outlines how a simulation running on a Carla server is controlled by one or multiple user scripts.
The screenshot in fig. 6.3 depicts an exemplary rural scene in Carla, involving two vehicles. It was recorded during an early development stage when the simulation client already supported to compute and render the occupancy grid corresponding to an ego vehicle's observations. Green cells are considered free, red cells are occupied and the state of blue

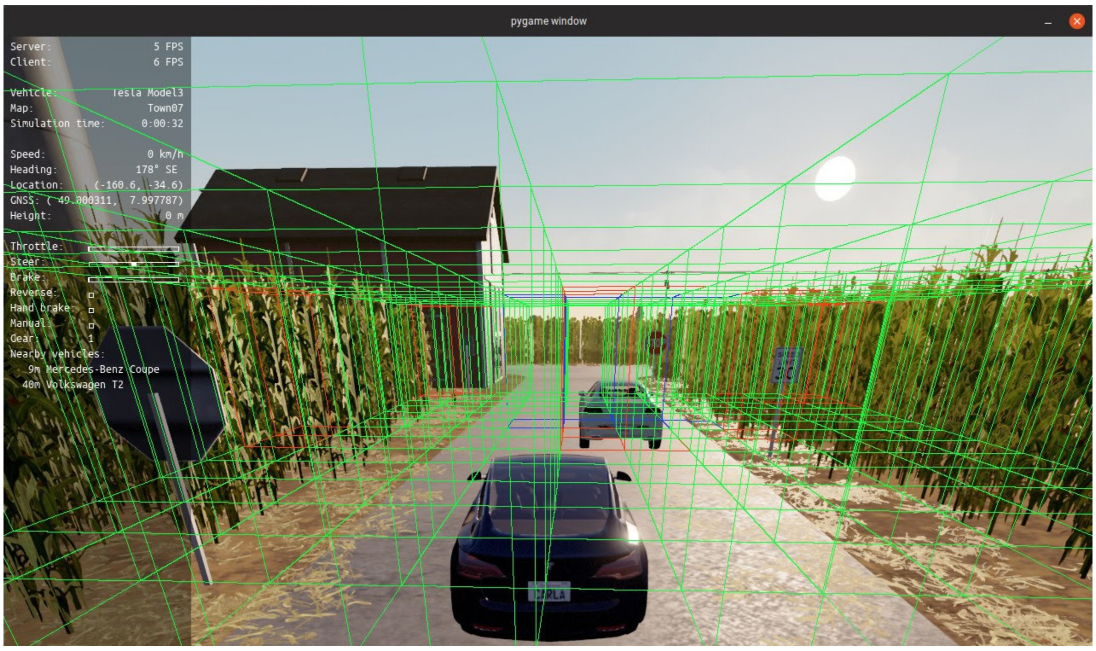
[11]`https://www.lgsvlsimulator.com`

[12]`https://unity3d.com`

[13]`https://github.com/ApolloAuto/apollo`

[14]`https://www.vectorzero.io/`

cells is unknown, e.g. due to sensor noise or because they are out of sight.



```
Server:                 5 FPS
Client:                 6 FPS

Vehicle:          Tesla Model3
Map:                    Town07
Simulation time:       0:00:32

Speed:                  0 km/h
Heading:               178° SE
Location:       (-160.6, -34.6)
GNSS: ( 49.000311,  7.997787)
Height:                    0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:        1
Nearby vehicles:
   9m Mercedes-Benz Coupe
  40m Volkswagen T2
```

Figure 6.3: Screenshot of an Exemplary Scene in Carla featuring two Vehicles and their Occupancy Grids

6.3 Server-Side Software Components

The following two sections discuss the implementation or integration process and respective details of the software components outlined in section 5.3.4, starting with server-side components. These constitute the central processing instance that all participants within certain geographical area connect to and exchange information with.

6.3.1 Message Broker

A message broker's responsibility as part of a publish-subscribe (or messaging-based) software system is to maintain connections to all clients and receive and re-distribute their messages according to certain rules. Usually this involves the notion of a *queue*, *topic* or *subject* as a basic routing mechanism for messages. Producers publish messages to a certain queue, which are then received by all consumers that had subscribed to that queue upfront. Figure 6.4 depicts this mechanism schematically.
A variety of messaging solutions and corresponding brokers already exist. When attempting to evaluate such, one has to distinguish between protocol and implementation. While the use of a messaging-based architecture has already been motivated in section 5.3.3,

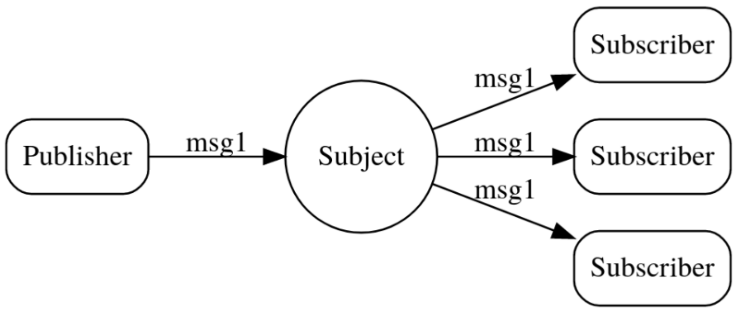


Figure 6.4: Schema of Publish-Subscribe Communication with Topics [Col19]

a concrete protocol and a corresponding software implementation need to be chosen in addition.
Publish-subscribe protocols for data streaming include AMQP, the Apache Kafka messaging protocol[15], MQTT, NATS[16] and more. Without conducting an in-depth evaluation of these alternatives, it can be concluded that each solution has various advantages and drawbacks with respect to different use cases. However, MQTT is the de-facto standard protocol for IoT applications and most widely spread. Accordingly, it has already proven effective and a variety of different proprietary and open-source implementations exist. Major benefits of MQTT include its low footprint in terms of messaging overhead and computational efficiency as well as the support for different quality of service (QoS) levels. QoS essentially allows the user to define the reliability of delivery for a certain message and has an impact on performance. Due to these benefits, MQTT is chosen as the pub/sub messaging protocol to be used for the present CP system.
In addition to deciding for a protocol, a corresponding message broker implementation needs to be picked. For MQTT, a variety of different implementations in different programming languages and with varying feature sets exist. The most common ones are Apache ActiveMQ[17] (Java), HiveMQ[18] (Java), Mosca[19] (JavaScript), Eclipse Mosquitto[20] (C) and RabbitMQ[21] (Erlang). A basic performance comparison between different brokers was conducted by [Müt19] and the results are depicted in fig. 6.5. Due to its good performance and easy setup procedure, Mosquitto was chosen for this work. However, since this thesis' implementation only uses standard MQTT features anyway, the broker could

[15]`https://kafka.apache.org/protocol`

[16]`https://nats.io`

[17]`http://activemq.apache.org/`

[18]`https://www.hivemq.com/`

[19]`http://www.mosca.io/`

[20]`https://mosquitto.org/`

[21]`http://rabbitmq.io/`

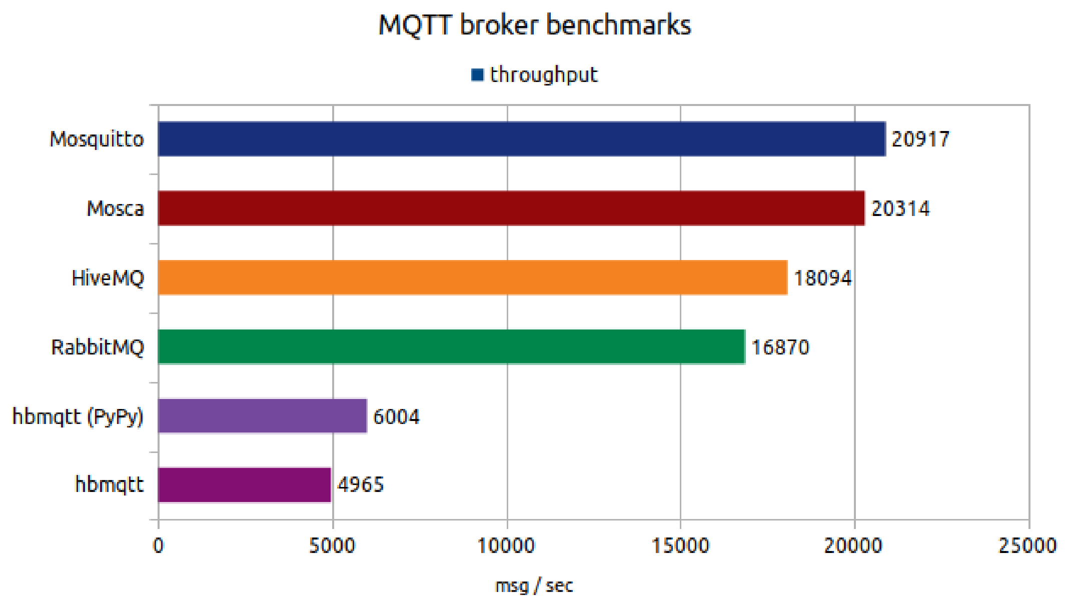


Figure 6.5: Basic Performance Benchmark of Different MQTT Brokers [Müt19]

easily be switched by any other one, as it is a modular, stand-alone software component.

6.3.2 Talky Fusion Node

The second server-side component, which is deployed "on the edge", is the fusion node. While the message broker only acts as an intermediary, the fusion node is a core component of the proposed CP system. It is responsible for the aggregation of all its connected clients' (vehicles, etc.) observations. It is realized as a stand-alone software component and, in a first iteration, using the Python programming language. However, after some initial evaluations it turned out that Python as a dynamically typed, interpreted language was too slow to perform fusion for multiple clients at a reasonable update rate. As a consequence, a second iteration re-implemented the fusion node in Go, which yielded a performance improvement of up to two orders of magnitude. Due to a high degree of modularity, encapsulation and separation of concerns (cf. [Mar17b]) the rewriting could be done with comparatively low effort.
The program relies on third-party open-source libraries with MIT-, BSD-3- and EPL-1.0 licenses.

The fusion algorithm as a core part of this component is modelled after the formal definitions described in section 5.4 and implemented to use multi-threading for parallelization, facilitated by Goroutines[22] and Go channels. Messages are received via MQTT from the

[22]https://tour.golang.org/concurrency/1

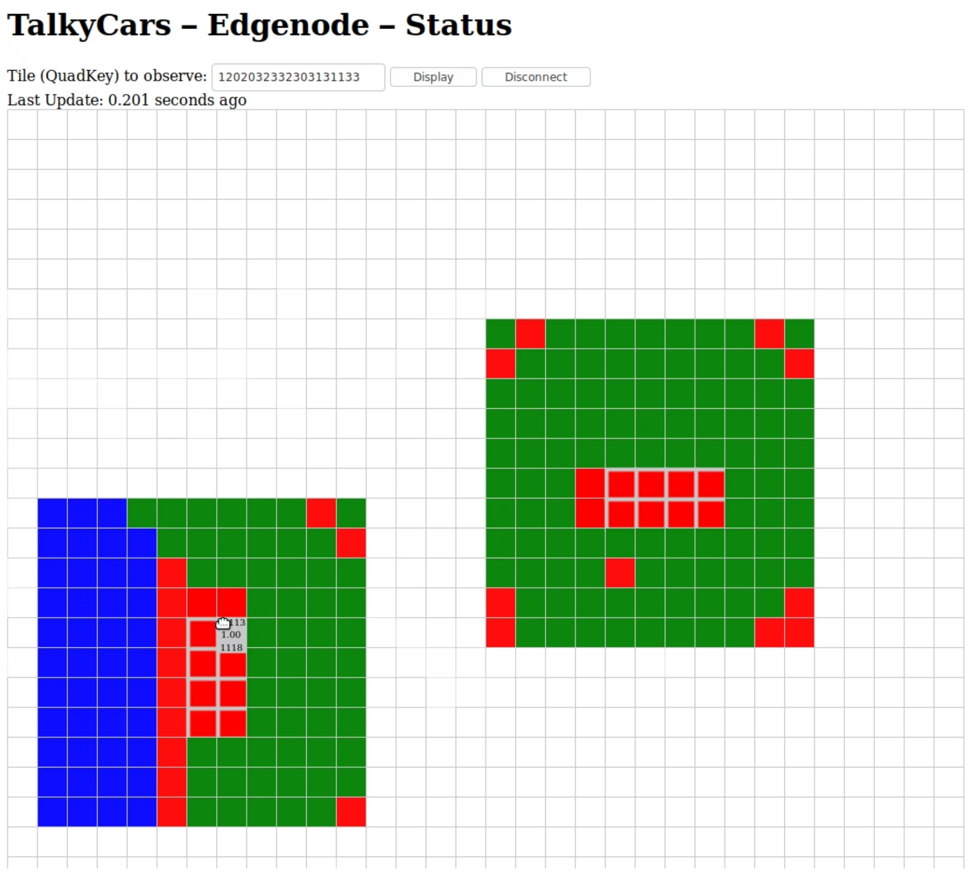
message broker using a subscription to a single topic, which all incoming observations are published to. Received messages are temporarily stored in an internal queue, from which they are picked up during the next fusion iteration. In line with the principle of periodic push, the fusion algorithm is executed at a configurable constant rate, e.g. 10 Hz. As part of an iteration, it fetches all observations from the queue, which are not older than a certain configurable threshold, e.g. one second. In a fusion procedure of $O(n*m*k)$ time complexity ($n \approx |cells|, m \approx |observers|, k \approx |observations\ per\ observer|$), the occupancy grids are merged into a single one, which is subsequently split into type 2 tiles (see section 5.3.2) and published to their respective MQTT topics periodically.

For instance, assume a vehicle is at position `120203233230313123011210`, $\lambda_2 = 19$ and $\lambda_3 = 16$. In that case, the current fusion node will be configured to be responsible for `1202032332303131` and the the vehicle will be subscribe to all of its surrounding level 19 tiles, including `1202032332303131230`. For this tile, among others, it will receive fused grids that were previously – ideally only few milliseconds ago – published by the fusion node.

As mentioned earlier, the fusion node is implemented as a stand-alone software component contained in a single executable file and can be executed as such without additional dependencies. Required program arguments include the message broker's address and port and the type 3 QuadKey of the tile, which this node is supposed to be responsible for. Further parameters, including the maximum observation age, the fusion rate, MQTT topic and QoS and more can be specified via an additional configuration file.

6.3.3 Web Visualization

This component was not presented in section 5.3.4 as part of the system as it is only used for development and debugging. Its purpose is to visually depict the output generated by the fusion node. Thanks to a uniform interface and a common message format specification, this visualization component could be easily implemented as a light-weight, stand-alone software component using Python. It subscribes to a desired MQTT topic to receive fused PER model instances and relays these occupancy grids to a web-frontend via a featured webserver using Websockets, where they are eventually displayed. The screenshot in fig. 6.6 depicts the respective visualization for a scene featuring two participant vehicles within the `12020323332303131133` type 2 tile. Green cells are considered free, red cells are occupied and blue cells are out of range or can not be observed for other reasons.


TalkyCars – Edgenode – Status

Tile (QuadKey) to observe: 1202032332303131133

Last Update: 0.201 seconds ago

Figure 6.6: Screenshot of an Occupancy Grid's Web Visualization

6.4 On-Board Client-Side Software Components

This section concerns with the implementation of all on-board software components, i.e. modules, which are usually run on an observer device. Usually, such observers will be vehicles, but might be road-side cameras, traffic lights or potentially even pedestrian smartphones as well. All client-side software is implemented in Python and relies on third-party open-source libraries with the following licenses: Apache 2, BSD-3, EPL, LGPL, MIT, MPL.

6.4.1 Simulator Bridge

This first component to run on the client-side, i.e. usually on a vehicle, is specific to this work, as it constitutes a software interface between CP system and simulation environment. In a real-world scenario, a similar component with (ideally) identical interfaces will exist nevertheless. However, it would wrap different functions. Instead of performing API calls to Carla and related data pre-processing it would rather communicate with either the previous AD pipeline module or the vehicle's sensory directly.

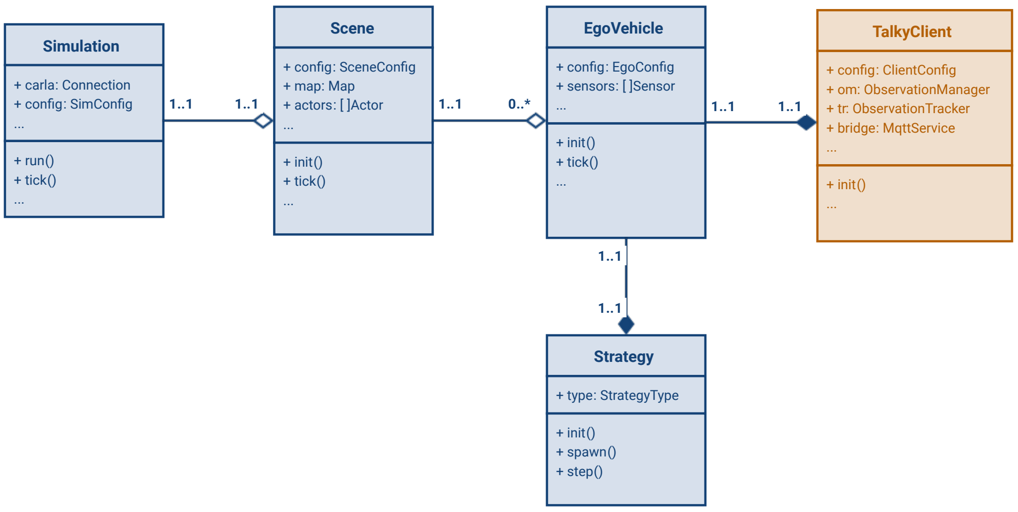


Figure 6.7: UML Class Diagram for Simulation Client

As Carla offers a rich Python API, this component is implemented in Python. As opposed to constituting a stand-alone component, however, it is integrated into the Talky client (see section 6.4.2) as a collection of encapsulated modules and classes. Following the guidelines for good software architecture presented in [Mar17b] and utilizing appropriate design patterns [Eri13], the implementation focuses on establishing a **strict boundary** between simulation client and Talky client, to make both possibly reusable.
The simulator bridge (or simulation client) communicates to the simulation server via a TCP socket connection and is responsible for performing multiple tasks in interaction with the server:

- Offer an interface to control the simulation environment, i.e. frame rate, map, weather conditions, other traffic participants, etc.
- Offer an interface to control an ego vehicle
- Receive and display rendered images from simulation server
- Receive sensor data (via a push mechanism, with a frequency synchronized to the server's tick rate) from the simulation server
- Pre-process sensor data
- Infer control commands according to a specified behavior or policy

Figure 6.7 depicts core sub-modules of the simulation client. In order to achieve a high degree of **modularity** and keep **clean code boundaries** between simulation-related code and the actual CP system, different abstractions are made.

The `Simulation` class can be thought of as an entry point to the program. It maintains connection and communication with the Carla server, displays the rendered images and synchronizes client- and server tick rate. Every simulation is initialized with exactly one `Scene`. It loads the map environment and spawns and controls all “non-player character“ (NPC) traffic participants, e.g. pedestrians. In addition, it holds references to one or more `EgoVehicle`s. Each of these represent a connected, intelligent car in the sense of autonomous driving. An ego vehicle, in turn, has access to several sensors, whose data it processes as well as actuators used to control the car. A given behavior, in form of a `Strategy` instance, specifies how the car interacts with its environment and what “decision“ it is supposed to take in which situation. Eventually, every ego vehicle holds an instance of exactly one `TalkyClient`, which corresponds to the respective component presented in section 5.3.4. All client-side cooperative perception tasks are performed inside this component and it acts as a bridge to the server-side subsystem via an intermediate MQTT connection with the message broker.
This modular structure enforces a strong **separation of concerns** and enables for distribution. That is, a certain scene might be hosted on one physical machine, while the contained ego vehicles run on another, e.g. to simulate latency or to better distribute computation load.

6.4.2 Talky Client

As mentioned before, this component is one of two core parts of the presented cooperative perception system, together with the `TalkyEdgeNode` (see section 5.3.4). It is implemented in Python and realized as one coherent application together with the simulation client. However, a highly modular structure is followed, so that different parts of the application are re-usable and can be executed separately. Responsibilities of this component include to:

- ... compute the current occupancy grid’s structure every frame.
- ... perform object detection based on received sensor data to derive occupancy states every frame.
- ... perform object tracking based on received sensor and derived occupancy states every frame.
- ... construct a PER model instance every frame.
- ... communicate its local state representation with the server-side fusion at a fixed rate.

Particular emphasis is laid on the tasks of detecting and tracking an object in the following. In order for a vehicle to construct an occupancy grid model or driveability map, detailed information on all surrounding obstacles is needed. Such include dynamic obstacles, like other traffic participants as well static obstacles, like curbstones, ground-mounted traffic signs, buildings, etc. The first step is to recognize them, also referred to as detection in the following, and a second step is to match and track them. That is, a detected object in one iteration or frame needs to be associated with an instance of itself in subsequent frames. Both problems are subject to current research and only covered in a very basic manner here.

While Carla offers a variety of sensors, including camera and LiDAR, unfortunately, there is no mechanism to directly retrieve all static and dynamic obstacles from the simulation[23]. Thus, they need to be derived from actual sensor data. For the task of **detecting** an obstacle, a basic heuristic is applied to perform **ray casting** on LiDAR sensor data. LiDAR data usually exists in the form of point clouds, which are, essentially, collections of 3-tuples of world coordinates. For every LiDAR ray, the respective coordinate refers to the point in space where it was reflected, i.e. hit an obstacle. Rays that are not reflected within the sensor's spatial range do not have corresponding triples in the point cloud. Given these data, the two-dimensional position of obstacles in Eucledian space – and thus of occupied cells – can be derived easily. Occupancy cells that do not have a LiDAR collision point falling into them are considered free. However, this indicator is only necessary, but not sufficient for a cell to be considered free. As mentioned in section 5.1.5, the state property is desired to be ternary rather than binary, i.e. distinguish between a cell being occupied or free on the one hand or having an unknown state on the other. Disregarding sensor noise, a cell within the observer's perception range is unknown exactly if no LiDAR ray has a chance to intersect or hit it. These are cells, which are hidden by an obstacle placed between the cell and the sensor. All cells that lie "behind" an obstacle, i.e. a cell that contains a LiDAR point, can be considered unknown. On the contrary, all cells "in front of" that obstacle can assumed to be free, as the LiDAR ray was able to pass them to eventually hit the target. To determine these cells, a ray casting algorithm is used to check for a ray's intersection with any of the "boxes" corresponding to cells between sensor and hit point. The estimation of a cell state can be summarized as follows, given a ray with vector direction $\vec{r_j}$ and magnitude d_j:

$$state(cell_i) = \begin{cases} occupied & \text{if } contains(cell\_box_i, \langle \vec{r_j}, d_j \rangle) \\ free & \text{if } intersects(cell\_box_i, \vec{r_j}) \\ unknown & \text{else} \end{cases}$$

[23]`https://github.com/carla-simulator/carla/issues/1832`

The implementation of *contains()* is trivial and for *intersects()* the ray casting algorithm shown in section B.2 was ported to Python. After Python's performance for this algorithm turned out to be comparatively poor, it was re-implemented in Cython[24]. Cython is an extension to the Python language and offers support to compile Python-like code to native C code and include it as a module. For computation intense algorithms, this usually improves performance dramatically. In the example of ray casting, the difference in performance between a Python- and C++ implementation of the exact same algorithm can be up to multiple orders of magnitude high [Nov17].
It is worth noting that, in a real-world system, object detection would usually be performed by a dedicated pipeline step. Normally, the Talky client would be supplied with already detected obstacles and their positions, which it can infer an occupancy grid from. Thus, the `OccupancyGridManager`'s matching function would normally take an object list as input, rather than a LiDAR point cloud. The current implementation could easily be adapted to support that by encapsulating the cell matching process in a further sub-module.

The second low-level fusion-related task mentioned above is **tracking**, which usually goes together with **matching**. However, for the sake of simplicity and in order to stay within the defined scope, an actual matching mechanism was not implemented. Instead, dynamic objects are identified by their IDs in the simulation. Cells are, trivially, identified by their position, i.e. their QuadKey. Basic tracking is based on the suggestion by [CD93], the authors of which stated that "*[n]ewly observed segments enter the model with a low confidence. Successive observations permit the confidence to increase, where as if the segment is not observed in the succeeding cycles, it is considered as noise and removed from the model. Once the system has become confident in a segment, the confidence factor permits a segment to remain in existence for several cycles*". Accordingly, a hash map is used to count the number of occurrences of a certain fact, e.g. the state of a specific cell or the presence of a certain car, in subsequent frames. The confidence of the corresponding relation is then set proportionally to its relative presence over the last few (e.g. 10) frames. If it was not seen over a certain number of subsequent frames, it is removed from the hash map and not "tracked" anymore. This rudimentary type of tracking was realized as a `LinearObservationTracker` sub-module and is the central source of confidence values as fourth parameter in the previously defined model's quadruples.

Communication with the server-side fusion node – via the message broker – is done inside the `SubscriptionManager` sub-module. On the one hand it acts as an interface for publishing an ego vehicle's current state. On the other hand it maintains according MQTT subscription for type 2- and type 3 tiles (see section 5.3.2), given the ego vehicle's

[24]`https://cython.org/`

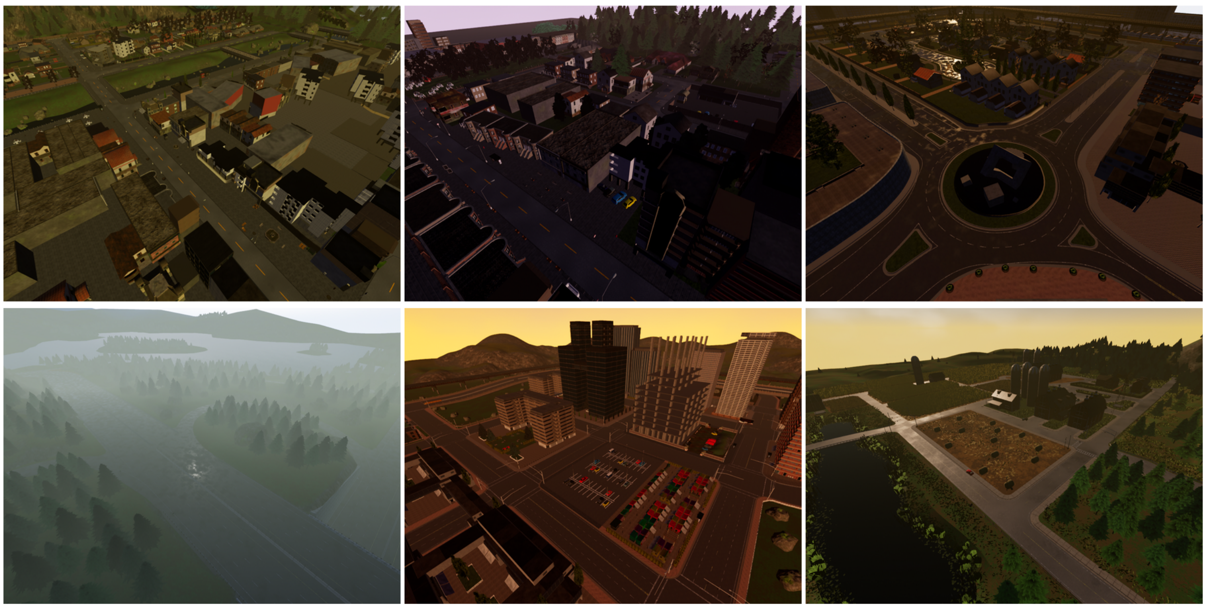


Figure 6.8: Screenshots of Carla Maps

current position. The component holds a reference to an instance of `MqttBridge` to handle low-level communication. Given this separation, MQTT could be easily switched out by a different publish-subscribe protocol.

6.5 Configurable Parameters

In complement to the previous sections, which provided an in-depth description of all involved software components, this section aims to give an overview over relevant parameters, that can be configured by the system's user. They can be categorized into three different classes, depending on which part of the system they affect.

Each of these parameters can be set either through a configuration file or as a program argument.

6.5.1 Simulation Parameters

The first set of parameters relates to the simulation.

Parameter	**Description**
`MAX_FRAME_RATE`	Upper bound for the frame rate which to run the simulation at
`SENSOR_{i}_POSITION`	Mounting position of the i-th sensor on the vehicle
`LIDAR_PPS`	Number of points per second sent by LiDAR sensor
`LIDAR_CHANNELS`	Number of vertical LiDAR channels
`LIDAR_ROTATION_FREQ`	Rotation frequency of the LiDAR sensor
`LIDAR_RANGE`	The LiDAR sensor's range

Table 6.1: Simulation Parameters

6.5.2 Scene Parameters

The second type of parameters is used to specify details about the current simulation scene.

Parameter	**Description**
`MAP`	Carla map or environment to run (see fig. 6.8)
`N_NPCS`	Number of "non-intelligent" vehicles present in the scene
`N_PEDESTRIANS`	Number of pedestrians present in the scene
`N_EGOS`	Number of connected, intelligent ego vehicles
`SPAWN_POINT_{i}`	Spawn point on the map for the i-th vehicle
`SPAWN_POINT_POLICY`	Alternatively: A policy for how to choose spawn points
`MAX_SPEED`	Maximum vehicle speed in $\frac{\text{km}}{\text{h}}$

Table 6.2: Scene Parameters

6.5.3 Cooperative Perception Parameters

The last category of parameters refers to such that specify certain aspects of the CP system itself.

Parameter	**Description**
`TILE_{i}_LEVEL`	Tile levels to use for geo partitioning
`MQTT_QOS`	MQTT quality of service
`OBSERVATION_MAX_AGE`	Maximum age (or “time-to-live“) for an observation to be included in fusion
`FUSION_RATE`	Rate at which to perform fusion (server-side)
`OBSERVATION_RATE`	Rate at which to publish observations (client-side)
`DECAY_FACTOR`	Exponential factor for temporal decay

Table 6.3: Cooperative Perception Parameters

6.6 Open-Source Contributions

The present project heavily relies on third-party open-source libraries without which the implementation would not have been possible in this form. Therefore, the efforts taken by the community are highly appreciated. Moreover, code contributions to several open-source projects have been made on the course of this thesis. Such include the Carla project[25], `buckhx/tiles`[26], a QuadKey implementation for Go, `buckhx/QuadKey`[27], a QuadKey implementation for Python (later `n1try/pyquadkey2`[28]), `ethlo/jquad`[29], a QuadKey implementation for Java and `johnnovak/raytriangle-test`[30], a benchmark of ray casting in different programming languages.

6.7 Summary

This chapter discussed details about the concrete implementation of concepts and components presented in chapter 5. The resulting software is subsequently evaluated in chapter 7 and constitutes an exemplary proposal for a novel, holistic cooperative perception system. In order to use it in real-world scenarios, one would have to add thorough quality assurance and testing as well as various performance optimizations. However, since the system is built following a modular approach it is comparatively easy to re-use or replace certain

[25]`https://github.com/carla-simulator/carla`

[26]`https://github.com/buckhx/tiles`

[27]`https://github.com/buckhx/QuadKey`

[28]`https://github.com/n1try/pyquadkey2`

[29]`https://github.com/ethlo/jquad`

[30]`https://github.com/johnnovak/raytriangle-test`

parts, while keeping others. Accordingly, the simulation client, for instance, might be switched out by a component interfacing with a real AD pipeline without too much effort. As this implementation is published as an open-source project[31], it can be used openly as a based for further research.

[31]`https://github.com/n1try/talkycars-thesis`

Chapter 7

Evaluation

After a comprehensive proposal for a cooperative perception system, including a novel modeling approach and an elaborate distributed software architecture, was presented in the previous chapters, this chapter aims to evaluate the system with regard to different aspects.

Section 4.3 presented a set of goals and requirements to be met by the proposed system. While previous chapters already discussed how most of them are individually addressed, a few demand for further investigation or empirical assessment. Accordingly, a two-fold evaluation is conducted. The first part concerns with evaluating the proposed **software architecture** in terms of performance, specifically with respect to the requirements of scalability (NF-C2) and efficiency (NF-C3). The second part of the evaluation aims to assess the system's qualitative **performance in cooperative perception** tasks, motivated by the overall goal of this thesis to facilitate the improvement of connected, autonomous vehicles' average perception quality (cf. section 4.3). Both parts are split into three sections each, that describe the respective goal and methodology, present the results and eventually discuss them in a brief conclusion.

7.1 Performance Evaluation

Section 4.3 stated the non-functional requirement for the system to be able to handle 202 concurrent network participants at minimum and to aim for low latency and on-vehicle load. The following evaluation thoroughly assesses the previously developed system with respect to both criteria, i.e. **system scalability** and **communication efficiency**.

7.1.1 Methodology

First, one or more metrics need to be determined with respect to which the evaluation of the above criteria should be conducted. With regards to scalability, a precise quantity is already given as a requirement. Namely, it refers to the number of concurrent clients (i.e. vehicles, pedestrians, etc.) the system is expected to handle at minimum. Assuming a fixed per-vehicle message publish rate – which is in accordance with the previously introduced *periodic push* principle – this translates to a **minimum number of observation messages (Q1)**, i.e. state representations, which the edge node must be able to process at a time. While the concrete message rate is a parameter that can be varied over the course of the evaluation, a hard minimum requirement of 202 concurrent vehicles is given. That is, assuming the entire system to operate at 10 Hz (i.e. both client- and server-side publish rate), the fusion node must reliably process 2020 observations per second without dropping below that rate. In addition, average **latency in milliseconds (Q2)** and average **message size in kilobytes (Q3)** per vehicle and per observation are to be determined to cover the communication efficiency aspect. Latency, in this specific context, refers to the average delay of a CP message until received by an ego vehicle and can generally be thought of the average "outdatedness" of a shared observation.

Message size (Q3) is constant per vehicle and can be determined trivially by inspecting and aggregating the individual sizes of incoming messages at the fusion node without any special setup. For better comparison, an additional boolean parameter `WITH_OCCUPANT` is introduced to denote whether or not an occupancy cell should include information about its potential occupant in addition to its pure state.

Concerning latency (Q2), the primary interest is to get insights about how it is composed. Instead of trying to estimate total latency as a function of traffic density / number of network participants, the focus is rather on getting insights about which parts of the fusion process are the most temporally critical ones to help later optimization of certain system components and functions. Accordingly, the relative durations $d_{i \in [0..6]}$ between relevant instants $t_{j \in [0..7]}$ of the fusion process, schematically depicted in fig. 7.1, are to be determined for a fixed parameter configuration. To help that, existing code is extended in various placed across Talky edge node and Talky client to add time measuring functionality.

In order to gather (Q1), a minimalist sub-system of the entire CP software system is employed. It still follows a client-server architecture and has the message broker and fusion node on one end and multiple (simulated) ego vehicles on the other. Since only quantitative measurements are of interest rather than the messages' actual content, a newly created message generator program is used to artificially simulate observations instead

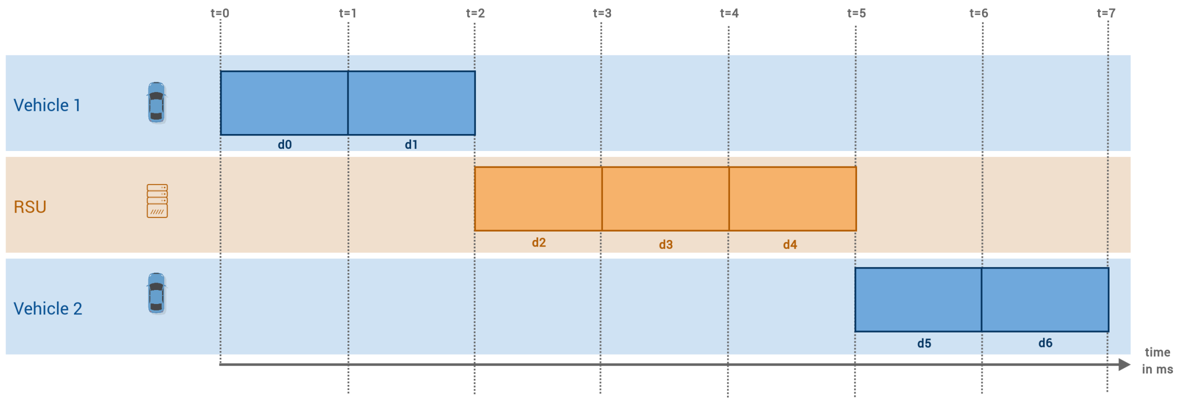


Figure 7.1: Timing Composition of Fusion Process

t = 1: An observation is obtained by local sensory and low-level fusion (including ray casting-facilitated cell matching, etc., cf. section 6.4.2)

t = 2: The observation is encoded locally, i.e. represented as a PER model and serialized to Protobuf format

t = 3: The observation is received remotely, i.e. at the fusion node

t = 4: The observation is decoded remotely, i.e. deserialized from Protobuf and converted to process-local data structures

t = 5: The observation is remotely fused with other relevant observations from different observers, encoded to a PER model instance and serialized to Protobuf again

t = 6: The fused observation is received locally by the ego vehicle

t = 7: The fused observation is decoded locally

of employing an actual Carla simulation with real Talky clients. Otherwise it would be infeasible to test with a large number of vehicles, due to exceedingly high computation load. The newly developed, multi-threaded message generator is parameterized with (1) the target size of the generated occupancy grid observations, (2) the number of concurrent clients to simulate and (3) a per-client frequency at which to push messages to the backend (see table 7.2). Moreover, supplementary code is added to the fusion node to track how often the `Publish()` method is called.

The test setup involves two physical machines, interconnected via a 1 $\frac{\text{Gbit}}{\text{s}}$ Ethernet network. One machine (AMD Ryzen 1600 3.2 GHz 6-core CPU, 16 GB RAM, Ubuntu 18.04) runs the backend part of the system, i.e. message broker and edge node, while the message generator is run on the other (Intel i5-6600K 4.5 Ghz 4-core CPU, 16 GB RAM, Manjaro

Parameter	**Value**
`TILE_1_LEVEL`	`24`
`TILE_2_LEVEL`	`19`
`TILE_3_LEVEL` (*)	`15`
`MQTT_QOS`	`1`
`DECAY_FACTOR` (*)	`0.11`
`FUSION_RATE` (*)	`10`
`OBSERVATION_MAX_AGE`	`3600`
`WITH_OCCUPANT`	`false`

Table 7.1: Constant Parameters of the Performance Evaluation. See section 6.5 for descriptions.

Paremeter	**Description**	**Values**
`N_EGOS`	Numbers of concurrent simulated clients exchanging CP messages	{`25, 50, 75, 100, 200, 400, 800, 1600`}
`OBSERVATION_RATE`	Frequency at which observations are periodically published to the network by each of its participants	{`1, 5, 10`}
`GRID_RADIUS`	Occupancy grid radius in number of cells. With $\lambda_1 = 24$ these are approx. equal to a grid size (i.e. observation range) of {`52.8, 100.8`} meters	{`11, 21`}

Table 7.2: Variable Parameters of the Performance Evaluation

18.1).

Table 7.1 lists all fixed parameters used throughout the entire evaluation. Parameters marked with a star (*) are mentioned for completeness, but should not have any influence on the measured quantity. For the experiment, randomly generated occupancy grids lie within the same type 3 tile are used in the observation messages.

Table 7.2 lists all variable, evaluation-specific parameters to be tested in order to get insights about their respective impact on the final results. To test all combinations of parameters for investigating (Q1), a grid search is conducted, i.e. a test is run for every combination in the Cartesian space of parameter values. For (Q2) and (Q3) only a single parameter set $\vartheta = \{\texttt{N\_EGOS} = 6, \texttt{OBSERVATION\_RATE} = 10, \texttt{GRID\_RADIUS} = 11\}$ is used instead.

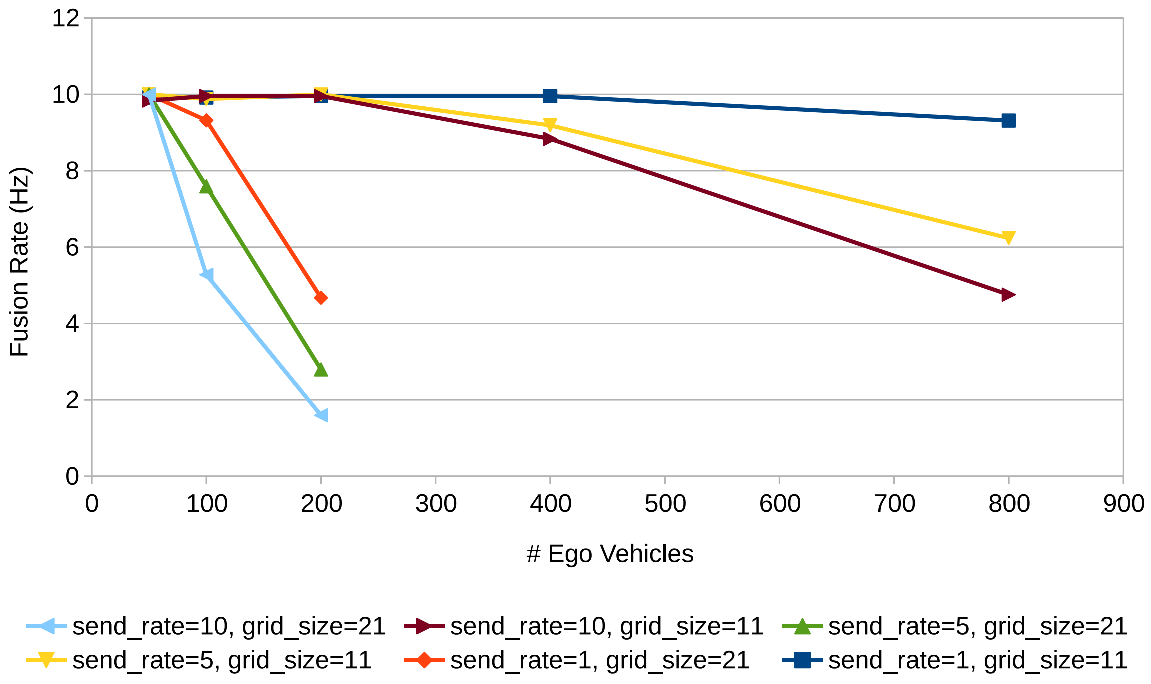


Figure 7.2: Average Measured Fusion Rate

In summary, two steps are performed to gather the above metrics. First, to find (Q1), multiple iterations – each according to one parameter set – are run using the minimal system subset in the presented two-machine setup. Second, the entire CP system including a Carla simulation instance is run in a single machine to obtain (Q2) and (Q3) for a fixed parameter set.

7.1.2 Results

In accordance with the structure presented in the previous section, results are obtained in two steps.

First, a set of experiments are conducted to get insights about the Talky edge node's **maximum fusion rate (Q1)**, depending on three different parameters. Whilst most parameters, including the edge node's fusion rate, are fixed to a certain value (see table 7.1), the number of concurrent network participants and their publishing frequency and grid size are varied. Consecutively executing the evaluation with a total of 30 different configurations yields the results depicted in fig. 7.2.
It can be seen that the maximum possible fusion rate heavily depends on both the incoming message rate and the occupancy grids' size. While the fusion node is able to maintain a rate of 10 Hz for up to nearly 200 concurrent vehicles if their observation range is $\sim$ 52.8 m (11 by 11 cells at $\lambda_1 = 24$), it tends to drop below that threshold as observa-

	GRID_RADIUS = 11	GRID_RADIUS = 21
WITH_OCCUPANT = false	12 kB	46 kB
WITH_OCCUPANT = true	32 kB	121 kB

Table 7.3: Average Measured Observation Message Sizes (occupied cells)

tion rate and grid size increase. Assuming a fusion rate of 10 Hz, the hard requirement of being able to handle ∼ 200 concurrent vehicles can only be fulfilled with the smaller observation range of `GRID_RADIUS=11`, regardless of the client's publish frequency. If the client vehicles are expected to publish their environment observations from a range of ∼ 100.8 m instead, the current Talky edge node implementation would not be able to keep up at a fusion rate of 10 Hz at all. However, if the system's fusion rate is lowered to, for instance, 5 Hz – which can still be sufficient for cooperative perception [TSG19] – the fusion node could fulfill the requirement for all test scenarios.
As an additional excursus, the MQTT broker's performance is evaluated separately to eliminate the possibility of it being a bottleneck to the system. Using the `mqtt-bench`[1] benchmarking tool it was found that Mosquitto (version 1.6.8) is able to handle up to 8000 messages/s with a message size of 12 kB (approximately corresponding to 11x11-cell grids, see table 7.3) and up to 4100 messages/s with a message size of 46 kB (21x21-cell grids). This means that the broker only becomes a limiting factor with more than 800 or 410 concurrent vehicles respectively. The benchmark results can be found in appendix section C.2.

In a second step, average message size latency are investigated. Results for the former are depicted in table 7.3. The minimum achievable **average message size (Q2)** of a Protobuf-serialized PER model instance was found to be 12 kB (46 kB) with a grid radius of 11 cells (21 cells), corresponding to an observation range of ∼ 52.8 m (100.8 m). Such minimalist model instances contain nothing but the observer information, the occupancy grid and an estimated state value for each cell. When including additional information about the occupant (vehicle, pedestrian, static obstacle, etc.) of a cell, the average message size increases accordingly.

As a result of investigating how an observation's total **latency (Q3)** is composed in the present CP system, the duration values presented in table 7.4 were observed. Timings shown in the table refer to those presented in fig. 7.1. As can be seen in the table, two separate, but related time series were measured for completeness. While the first includes an entire round trip, the second excludes $t = 0$ and starts at instant $t = 1$, i.e. at the moment a local observation is already present. With respect to the evaluation goal, this is more

[1]`https://github.com/takanorig/mqtt-bench`

Instant	**Time Offset** (with local perception)	**Time Offset** (without local perception)	**Duration** ($t_{i-1} \rightarrow t_i$)
t=0	0 ms	–	–
t=1	260 ms	0 ms	260 ms
t=2	288 ms	28 ms	28 ms
t=3	341 ms	81 ms	53 ms (3 ms)
t=4	343 ms	83 ms	2 ms
t=5	501 ms	241 ms	58 ms (8 ms)
t=6	545 ms	285 ms	44 ms
t=7	549 ms	289 ms	4 ms

Table 7.4: Cooperative Perception Latency Composition

Duration is additionally normalized with the expected delay of $\mu_{\delta t} = 50\,\text{ms}$ caused by periodic push at 10 Hz

meaningful, as it only measures the actual CP process and disregards observer-specific performance of local perception and low-level sensor fusion. In the present experiments, a globally fused observation is on average 289 ms old when it arrives at a client again. In other words, given the system runs at 10 Hz it takes 289 ms for a local observation to take the journey from a vehicle over the message broker and Talky edge node back to a vehicle. However, two key points must be kept in mind when interpreting these results. First, a local area network (LAN) with Ethernet was used instead of cellular 5G. Second, some of the depicted duration values include an "idle" delay caused by the fact that the system runs at a fixed rate (10 Hz in this case). For instance, when an observation is received at the fusion node, it takes, on average $\mu_{\delta t} = \frac{1}{2} * \frac{1\,\text{s}}{10} = \frac{1}{2} * 100\,\text{ms} = 50\,\text{ms}$ to get "picked up" by the fusion routine. After normalizing the duration values by this average delay (denoted in brackets in table 7.4), it can be seen that receiving an observation locally ($t_4 \rightarrow t_6 \hat{=} d_5$) takes longest, although receiving it remotely is much faster ($t_2 \rightarrow t_3 \hat{=} d_2$). This might potentially be caused by a "queuing" effect resulting from poor performance of the client-side message processing, but demands for further investigation in future work. Local encoding (model instantiation and Protobuf serialization) ($t_1 \rightarrow t_2 \hat{=} d_1$) makes up another significant part of the total latency, but is still faster than with the previous approach of using Cap'n'Proto (cf. section 6.1.2). Overall, the average total delay of 289 ms (189 ms normalized) can be considered acceptable, especially given that the current implementation is rather a proof-of-concept than an optimized system solution. However, further research is required to investigate latency in real-world scenarios and using actual cellular networks.

A further discussion about the implication of any of these results is done in the next section.

7.1.3 Discussion & Conclusion

With respect to scalability, it was shown that a single Talky fusion node is able to fulfill the requirement of handling more than 202 concurrent clients with an observation range of $\sim$ 50 m up to a fusion rate of 10 Hz. Given a 100 m observation range, the system would at least be able to support 5 Hz. While the authors of [CM17] suggest to run their system at 10 Hz, [TSG19] shows that certain optimizations can decrease the required message rate for cooperative perception to 4.5 Hz without sacrificing perception quality. Hence, the present prototype system's performance can be considered acceptable, although there is large room for optimizations. Such include the following.

- Algorithmic optimizations of the fusion routine: the current implementation has superlinear complexity and might potentially be reworked to scale better. For instance, Petrich et al. [PAKZ18] showed that a relational traffic scene representation – like the present PER model – can be transformed to a **tensor format**, which potentially enables for fusion to be represented as matrix operations. Such can be run on a GPU in a highly optimized fashion and potentially boost performance by orders of magnitude. Another potentially promising optimization is to extend the current, naive fusion to **probabilistic fusion**, which is a concept that, to the best of my knowledge, no previous work had covered, yet. The basic idea here is to sample a smaller subset of all available observations using a particular probability distribution, so that the entire spatial area is still best represented in the sample.

- Parameter optimization: for the sake of simplicity, only a very small set of different parameters were tested in the previous evaluation. However, for real-world deployment, one would want to thoroughly **determine optimal values** for any of the parameters presented in section 6.5. For instance, the authors of [GTW15] run their CP system at only 1 Hz and with a communication range of 300 m (compared to 611 m or 1223 m in the present experiments). This holds potential to dramatically improve performance.

- Message generation optimizations: [TSG19] presents a rich set of thought on how to **optimize the generation and publishing** of CPMs. For instance, the authors worked out sophisticated heuristics to determine when a message would contain enough valuable information about the current to be published and when it is better held back for another few time steps. In complement to that, [BM13] presents a way to efficiently **estimate the relevance** of a CAM for its recipient, mainly based on the sender and recipient's trajectory.

With respect to communication efficiency, the present evaluation found that, given a per-vehicle observation range of $\sim 50\,\mathrm{m}$, the average message size ranges from 12 kB without occupant information to 32 kB with such. Naively assuming an available bandwidth of 200 Mbit (cf. section 5.2.1), the network could handle $\sim$ 2080 or 780 messages per second respectively. Without conducting a more elaborate estimation, this can generally not be considered sufficient. Accordingly, the format of exchanged messages would have to undergo further optimization in the future. Such optimizations might include server- and client-side **caching** of de-facto constant properties of actors, e.g. the `boundingBox` of a `DynamicObstacle` used within an `observedBy` relation of an `OccupancyGrid` (see section 5.1.5). In addition, the cells of an occupancy grid can be represented more efficiently by only transmitting the top-left and bottom-right cell's hash, instead of all, since that information is redundant. Moreover, the optimization methods (e.g. with regard to message generation) mentioned above apply to improving communication efficiency likewise.
As a result to investigating a CP iteration's total latency and how it is composed, an average "outdatedness" of shared observations of $\sim 289\,\mathrm{ms}$ was found, when disregarding the local measuring process. Future optimizations might minimize the idle delay caused by the system running at a fixed rate and therefore further reduce the total delay to 189 ms. Whether or not this is a suitable delay for a CP system is discussed in the next section 7.2.

7.2 End-to-end Evaluation

This part of the evaluation analyzes the proposed system's cooperative perception performance in and end-to-end fashion. Goal is to determine to what degree autonomous connected cars can improve their perception quality through cooperatively sharing their local belief about the environment. While the performance evaluation in section 7.1 examined certain characteristics of the system itself, in this part it is rather viewed as a black box to measure its impact end to end.

Different approaches can be taken to measure overall CP performance. For instance, the authors of [LKC+13] evaluate their system with regard to planning. More specifically, they compare the smoothness or "jerk" of an automated vehicle's generated trajectories with and without cooperation. Another way is presented in [CTYF19]. In this case, the authors evaluate their low-level fusion-based system with regard to detection range and confidence, i.e. they measure the percentage and distance of detected obstacles alongside the average observation confidence with and without CP respectively. The approach taken by the present work is similar to the latter one and focuses on perception / detection rather than on planning. Given multiple automated, connected cars in a simulation, goal is to observe the **overall average perception quality improvement** in terms of detection accuracy and confidence. As a part of that, it is also investigated how this potential improvement is impacted by the **observer network's density** and different parameter values for **temporal decay**.

7.2.1 Methodology

7.2.1.1 Idea & Structure

The present evaluation is structured into three closely related parts. First, the proposed system's **general suitability** (**part 1**) for cooperative perception tasks is assessed, i.e. goal is to determine the extent to which perception quality can be improved through cooperation. Subsequently, it is further investigated how this potential improvement is affected by the **number of connected, communicating vehicles (part 2)** and by different **temporal decay factors (part 3)** (see section 5.4.5). As mentioned earlier, the evaluation is conducted in an end-to-end fashion in the sense that measurements taken in multiple simulated scenarios are accumulated and viewed as a whole, rather than looking into particular aspects of the system itself. The setup is as follows.

A variable number of ego vehicles are simulated in a pre-defined Carla environment. Each of these comprises an instance of the client-side parts (cf. section 5.3.4) of the previously developed CP system and is connected to a centrally deployed instance of the respective server-side part, i.e. message broker and fusion node. Given pseudo-randomly **generated start- and destination** waypoints on the Carla map, the agents (ego vehicles) automatically traverse their environment while facing pseudo-randomly placed static and dynamic obstacles, such as other traffic participants. While doing so, their observations in the form of PER model instances, each of which contains an occupancy grid and respective cell state estimations, are recorded for later analyses. Multiple different scenarios are consecutively tested, in each of which static obstacle positions and start- and destination points of dynamic obstacles and ego vehicles are varied. Every configuration is run multiple times to later calculate an average score. The key point of this evaluation is to "virtually" **run each set of experiments twice** in addition: once with the Talky fusion node **turned on and off** respectively, i.e. with the ego vehicles communicating with each other or not. Subsequently, both result sets are compared to learn whether or not CP improves perception quality.
For illustration purposes, fig. 7.3 schematically shows an exemplary Carla scene (*Town01*) from a top-view perspective as it might be used in this part of the evaluation. It involves three ego vehicles (green boxes) driving in a straight line, several stationary or moving other vehicles (blue boxes) and several pedestrians (not shown).

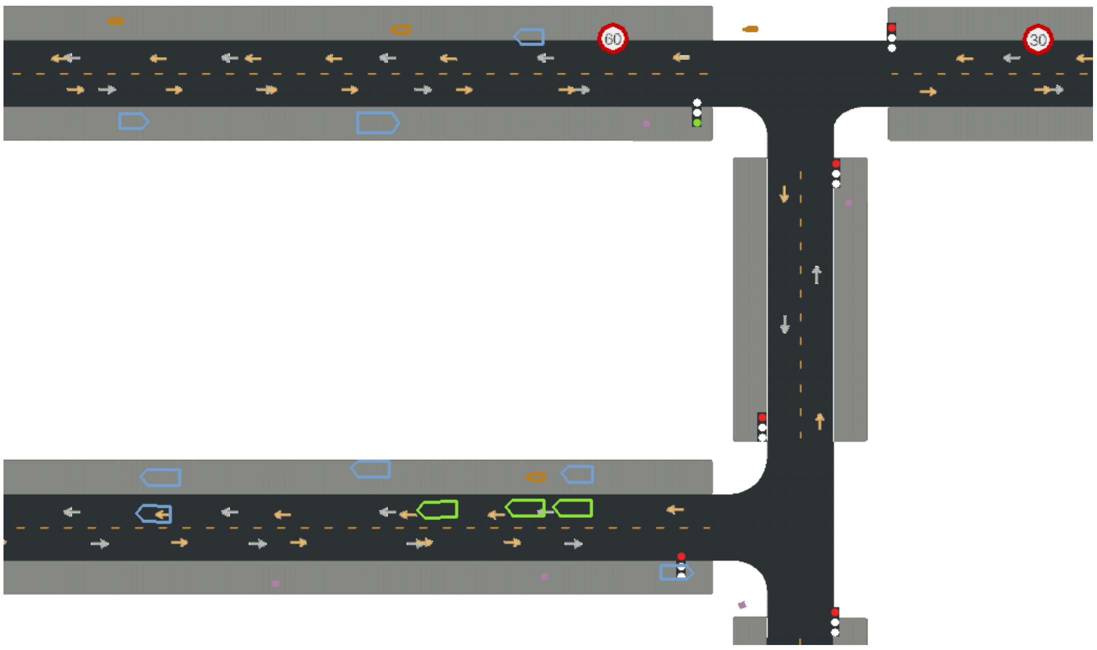


Figure 7.3: Schematic Top-View of an exemplary CARLA Simulation Environment used for Evaluation (*Town01*). Green boxes indicate ego cars, blue boxes correspond to other, non-“player“ vehicles.

7.2.1.2 Parameters

For all following tests, the fixed parameter values listed in table 7.5 are used. In addition to these, three further parameters are involved, which are varied across the three different test sets accordingly.

- `N_VEHICLES` (p_1): The number of (communicating) ego vehicles to be used in an experiment. Fixed to $p_1 = 6$ for parts 1 and 3, varied wihtin $p_1 \in \{1, 3, 6\}$ for part 2.
- `DECAY_FACTOR` (p_2): The factor to be used for decaying (“down-weighing“) older observations during fusion. See λ in eq. (5.6). Fixed to $p_2 = 0.14$ for parts 1 and 2 and varied within $p_2 \in \{0.05, 0.08, 0.11, 0.14\}$ for part 3.
- `SEED` (p_3): Random seed used to initialize pseudo-randomness for reproducible start- and destination point generation. Fixed to $p_3 = 4$ for part 3 and varied within $p_3 \in \{4, 8, 16\}$ for all other experiments.

In each of the following three experiments, every scene setup is run five times for every parameter combination. That is, for the first part, $k_1 = |range(p_3)| * 5 = 15$ tests are instantiated. Analogously, $k_2 = |range(p_1)| * |range(p_3)| * 5 = 45$ and $k_3 = |range(p_2)| * 5 = 20$ runs are performed for parts 2 and 3 respectively.

Parameter	Unit	Value	Parameter	Unit	Value
`MAX_FRAME_RATE`	fps	30	`SPAWN_POINT_POLICY`	-	random
`LIDAR_PPS`	pps	4000	`MAX_SPEED`	km/h	25
`LIDAR_CHANNELS`	-	3	`TILE_1_LEVEL`	-	24
`LIDAR_ROTATION_FREQ`	Hz	30	`TILE_2_LEVEL`	-	19
`LIDAR_RANGE`	m	48	`TILE_3_LEVEL`	-	15
`MAP`	-	Town01	`MQTT_QOS`	-	1
`N_NPCS`	-	6	`OBSERVATION_MAX_AGE`	ms	2000
`N_PEDESTRIANS`	-	90	`FUSION_RATE`	Hz	10
`N_STATIC`	-	75	`OBSERVATION_RATE`	Hz	10

Table 7.5: Constant Parameters of the Perception Evaluation. See section 6.5 for descriptions.

7.2.1.3 Data Collection

As mentioned before, every ego vehicle records both its local (M_i^{loc} in section 5.4.6) and – in the case of CP being enabled – its received fused observations (M^{glob} in section 5.4.6). Observations are dumped to a file to be then used in a subsequent, offline (i.e. after the simulation has finished) analysis. **Offline analysis**, as opposed to computing evaluation scores online and in "real time" during the simulation, is necessary out of performance reasons, as these computations would dramatically slow down the simulation. Alongside the ego vehicles themselves, an additional small program, the *DataCollector*, is run during the simulation, that is responsible for retrieving the actual, true obstacle position from the simulator. These are recorded to a file as well to be later used as ground truth measurements for comparison in the analysis. An important thing to note is that, out of technical reasons, only *occupied* cells are recorded, but not *free* ones. Accordingly, this evaluation only considers **true positives** and **false negatives** with regard to *occupied*. This means that it can be distinguished between an occupied cell being observed correctly or not, while cells, that are *free* in the ground truth data, are not included in the score.

7.2.1.4 Data Analysis & Scoring

The actual data analyses are subsequently performed in a downstream step using a separate Python script, the *DataEvaluator*. It first reads in all observations from all ego vehicles over all time steps of the simulation (i.e. the time it took for all ego vehicles to reach their destination) plus the previously mentioned ground truth data collector for a particular run.
Subsequently, the script essentially compares ground truth data to observations; once to

those with CP enabled and again to the local ones only. Doing so, all cells (= type 1 tiles, cf. section 5.3.2) that are contained in the observer's current type 2 tile as well as in any of the neighboring type 2 tiles are considered. Figure A.2 in appendix section A.4.1 schematically depicts this in greater detail. However, as mentioned earlier, only the state of an (occupied) occupancy cell will be considered, so an observation corresponds to a true or estimated cell state.

As a result of those comparisons, two scoring metrics – **recall (REC)** and **mean-squared error (MSE)** – are computed for the aggregated set of pairs of true and estimated cell state for the entire run. The meaning and definition of REC and MSE with regard to the current context is briefly explained in the following.

Let $\vec{y_i}$ be a three-dimensional vector representing a cell's ternary (*free*, *occupied*, *unknown*) state confidence, on which only one entry can be non-zero at a time. For instance, a 30 % confidence for a cell state being *occupied* could be expressed as $\vec{\hat{y_i}} = (0, 0.3, 0)$. Further, let a distinction be made between the true state vector $\vec{y_i}$ and its estimation (as part of a vehicle's observation) $\vec{\hat{y_i}}$ and let Y and $\hat{Y}$ be the sets of all such true- and estimated state vectors respectively. Then, the following definitions for total, accumulated recall and MSE can be given.

$$MSE(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \vec{y_i} \cdot (\vec{y_i} - \vec{\hat{y_i}})^2 \tag{7.1}$$

$$REC(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} -sgn(\vec{y_i} \cdot (\vec{y_i} - \hat{y}_i) - 1) \tag{7.2}$$

Example: Assume a grid that consists of only two cells, the first of which is occupied and the second is free. Moreover, consider two observers, each estimating the grid cells' states for exactly one time step. In the example, $\vec{\hat{y}}_{i,j}$ denotes the j-th vehicle's observation of cell i, i.e. $\vec{\hat{Y_i}}$ are all observations for cell i. The matrices' first (second) columns are the first (second) vehicle's observations.

$$\vec{y_1} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \vec{y_2} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \hat{Y_1} = \begin{pmatrix} 0 & 0 \\ 0.8 & 0.6 \\ 0 & 0 \end{pmatrix}, \hat{Y_2} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0 \\ 0 & 0.9 \end{pmatrix}$$

$$MSE(Y, \hat{Y}) = \frac{1}{4}(\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0.2 \\ 0 \end{pmatrix}^2 + ... + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ -0.9 \end{pmatrix}^2) = \frac{1}{4}(0.2^2 + ... + 1^2) = 36.25\%$$

$$REC(Y, \hat{Y}) = ... = \frac{1}{4}(-sgn(0.2 - 1) - ... - sgn(1 - 1)) = \frac{1}{4}(1 + 1 + 1 - 0) = 75\%$$

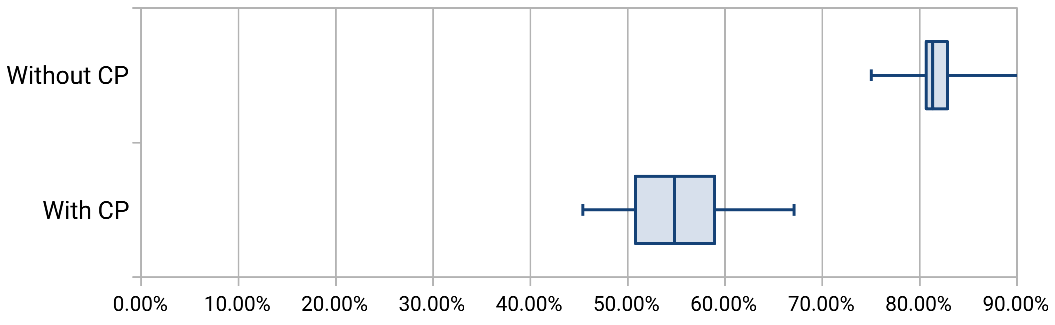
7.2.1.5 Test Setup

As a test setup, the same physical machines as in section 7.1.1 are used. The first machine runs the Carla simulator and all backend components, i.e. message broker and Talky fusion node. For the configurations comprising only one or three ego vehicles, all client-side components, i.e. simulation client and Talky client, are run on the second machine. In the last configuration involving six ego cars, four of them are run on machine 2 and the remaining two out of six on machine 1.

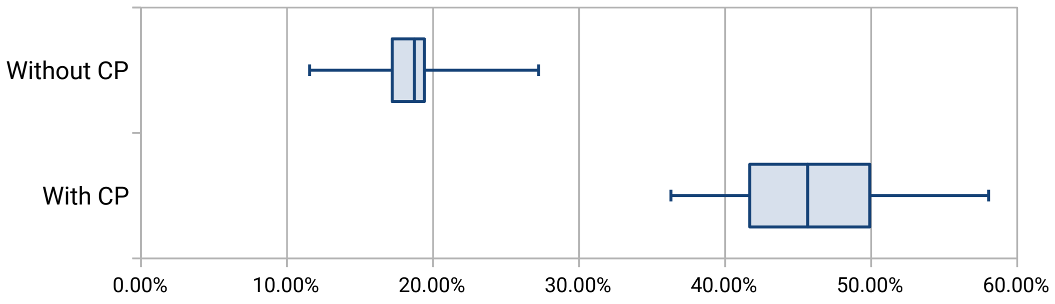
7.2.2 Results

It was mentioned before that the first step involves evaluating whether perception quality can be improved through the proposed CP system. Therefor, different simulation scenes were instantiated and repeatedly run for a total of 15 experiments. Given the recorded data, CP was “virtually“ turned on and off during evaluation to measure the difference with respect to MSE and recall. This yielded the results depicted in fig. 7.4.
It can be clearly seen that both scores are better with cooperative perception, i.e. with the ego vehicles communicating and exchanging their environment state representations. Looking at the different in arithmetic mean for recall and MSE in the above scenario, it can be found that $\Delta\varnothing_{MSE} = -27.13\%$ and $\Delta\varnothing_{REC} = 27.73\%$. In addition, when looking at the average number of unknown cells, that could potentially have been observed, one finds that $\Delta\varnothing_{unknown} = -41.11\%$, i.e. ~ 40 % more cells are “unveiled“ for each vehicle over the course of the simulation with CP.

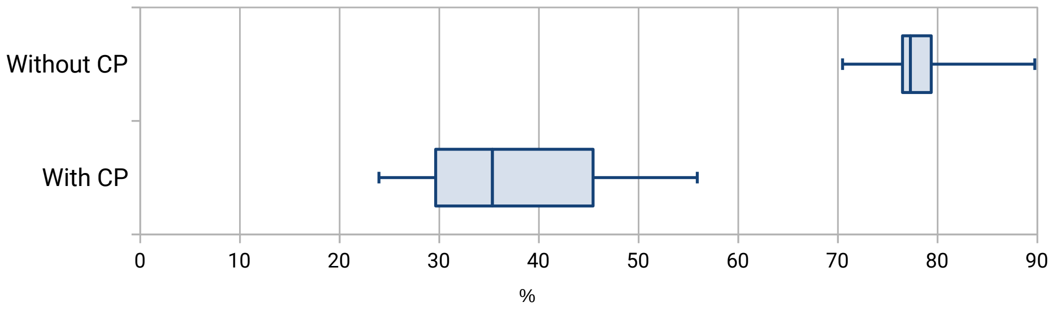
After an actual improvement in perception quality through CP could be observed, the following additional results were obtained from viewing that improvement in relation to two parameters, the number of employed connected vehicles and the temporal decay factor. Since the absolute recall and MSE values for two different configurations of `N_VEHICLES` can NOT be compared to each other for various reasons, only relative changes are considered in the following. In other words, given exemplary values of $MSE_{n=6,CP}(Y_1, \hat{Y}_1) = 50\%$ and $MSE_{n=3,CP}(Y_2, \hat{Y}_2) = 45\%$ does NOT imply that six vehicles are 5 % better than three. Conversely, one can instead draw such a conclusion when comparing relative differences between with and without CP, e.g. $\Delta\varnothing_{MSE,n=6} = -10\%$ and $\Delta\varnothing_{MSE,n=2} = -5\%$.
Figure 7.5 (left) shows the results for different configurations when varying the number of employed connected vehicles, or observers, in general. First, it becomes clear that there is no quality improvement at all between CP being turned on and off when using only one vehicle. This is plausible, since the key point of cooperative perception is to benefit from other traffic participant’s observations in addition to one’s own. Moreover, no significant increase or decrease in MSE can be perceived when varying the number of network participants between three and six. With regard to the temporal decay factor, a



(a) Average MSE for `N_VEHICLES = 6` (lower is better)

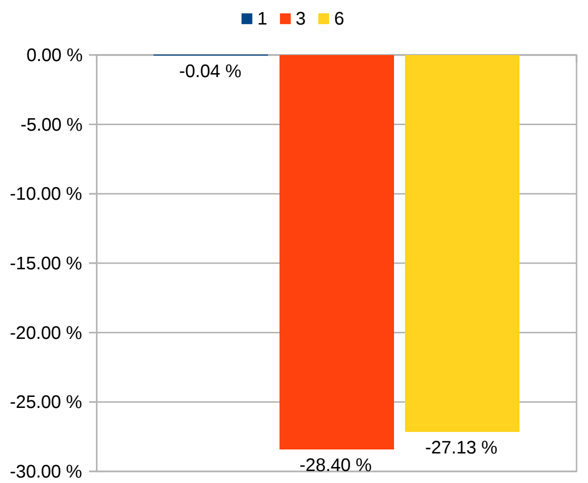


(b) Average recall for `N_VEHICLES = 6` (higher is better)

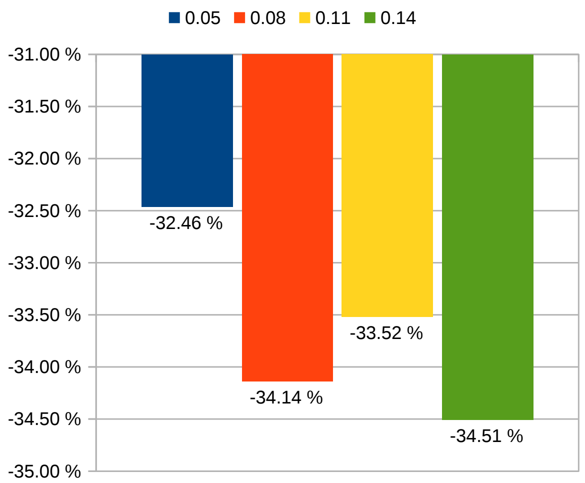


(c) Average percentage of *unknown* cells for `N_VEHICLES = 6` (lower is better)

Figure 7.4: Perception Evaluation Scores – Part 1



(a) Average MSE change due to CP for different `N_VEHICLES`



(b) Average MSE change due to CP for different `DECAY_FACTOR`

Figure 7.5: Perception Evaluation Scores – Part 2

value of 0.14, i.e. a comparatively "aggressive" down-weighing of old observations, seems to yield the best results. However, since the differences are only marginal, one might not speak of a significant impact of `DECAY_FACTOR` on the final score, at least in these brief experiments.

7.2.3 Discussion & Conclusion

The previous evaluation clearly unveiled that the average, overall perception quality can be improved through cooperation, specifically through the use of the presented CP system. Similar to what Chen et al. [CTYF19] discovered for their CP system, it was found that both a vehicle's perception range or field of view as well as the average confidence for single measurements can be increased. In the present experiments, an average improvement in total, accumulated mean squared error of $\sim 27\%$ and a $\sim 27\%$ better recall were discovered likewise. In addition, it was found that $\sim 41\%$ more potentially observable cells were actually observed, i.e. they were assigned a state estimation other than *unknown*. As a result of briefly investigating the total MSE improvement as a function of the number of connected vehicles and the decay factor used for fusion, no significant impact of neither of both was found.

While the general suitability of the proposed system to increase overall end-to-end perception quality was experimentally shown, further evaluation with a greater range of different scenarios and parameter settings is needed in order to gain more extensive insights about conceptual strength and weaknesses and about limitation of the current prototype implementation. Presumptions are that the perception quality improvement gets even more significant in larger-scale scenarios and using a highly-optimized, low-latency implementation.

7.3 Summary & Conclusion

Previous sections aimed at evaluating the system proposed in the context of this work with regard to two different aspects, namely software performance and end-to-end perception quality.
First, the system's performance was analyzed with respect to three different metrics. These include the maximum number of concurrent network participant the network is able to handle per area, the average message size sent over the network and the average latency or delay of an observation on its pass through the system. Multiple analyses revealed that the minimum requirements to these metrics, stated in section 4.3, can be fulfilled by the current prototype implementation. However, there is large room for optimizations to consider for a real-world deployment of the system.
Second and last, an end-to-end evaluation was conducted to measure the system's general suitability for cooperative perception tasks, i.e. whether it can help to increase overall perception quality. Therefor, three different sets of experiments were conducted in a simulation, all of which employed an entire instance of the proposed system. It was found that overall perception quality can be improved by about 27% on average and thus the system is capable of fulfilling its core purpose in principle. However, this is only known to hold true for comparatively few vehicles driving in a relatively minimalist scenario. Additional experiments are desirable to be conducted to gather further evidence.

Chapter 8

Conclusion & Future Work

This last chapter aims to conclude the present work. For this purpose, a summary of the contents and results of previous chapters is given first, followed by an outlook on potential future work to complement this thesis.

8.1 Summary

Overall goal of this work was to design and implement an end-to-end concept for a cooperative perception system. Considered aspects range from environment modeling and state representation over communication- and software architecture throughout to sensor fusion and an in-depth evaluation.

After this goal was motivated in chapter 1 and required background knowledge and fundamentals were provided in chapter 2, chapter 3 presented the current state of the art in all relevant research topics and differentiated the present approach from existing solutions.

Chapter 4 aimed to analyze current approaches and their individual limitations and presented a comprehensive set of goals, functional and non-functional requirements for a novel cooperative perception system to address these limitations. It was observed that current approaches usually lack comprehensiveness, standardization and universality as well as scalability. The latter is usually imposed by the usage of dedicated short-range communication in vehicular ad-hoc networks, which gave rise to employing fundamentally different communication technologies and patterns.

Chapter 5 was about elaborating an end-to-end concept that is based on novel techniques and aims to overcome previously described limitations. First, a comprehensive, extensible, yet not complete model for dynamic traffic scenes was proposed. It combines low-level attributes with high-level features and relational knowledge in a generic way and was designed to fulfill previously stated requirements. Geo tiling, occupancy grids and probabilistic entity relationship models were introduced as core building blocks. Second, major communication technologies and topologies were discussed and a detailed comparison between DSRC-based VANETs and client-server architecture utilizing cellular networks was conducted. Several advantages of the latter regarding latency, throughput and network utilization were outlined. Third, a holistic system architecture was designed in form of a distributed, messaging-based client-server software solution. Edge computing concepts were incorporated to reduce latency and distribute load. Resilience and scalability are facilitated by the novel approach of geographical partitioning. Eventually, a fundamental concept on how to perform high-level sensor fusion in the context of cooperative perception was developed. In accordance with the previously presented system design, it involves a centralized fusion node and employs the novel concept of doubly-updated merging.

Chapter 6 discussed details about the concrete implementation of the previously presented concepts. All involved software components were described and categorized into server-side components, including message broker and fusion node and on-board, client-side components, including Talky client and simulator bridge. The entire system was implemented in a modular way with clear boundaries and strict interfaces with the goal to enable for easy replacement of individual components, e.g. to use a different simulation environment or messaging backend. Moreover, a multitude of technology choices were made. Carla was chosen as a simulator, MQTT was picked as the central pub-sub messaging protocol for communication among different components and Protobuf was chosen to be employed as a highly efficient binary message format. Eventually, all relevant system parameters and their respective purposes and effects were presented. They were classified into simulation- scene and cooperative perception parameters.

In chapter 7 a two-fold evaluation was conducted to investigate software performance and scalability as well as the system's general suitability for cooperative perception tasks. It was found that the system is able to scale sufficiently and can meet the requirements, which were previously designed based on realistic assumptions and estimations of urban traffic volumes. However, further optimization steps are still recommendable in order to apply the system to real-world scenarios. With regard to qualitative performance, the evaluation revealed a potential improvement in overall perception quality of 27 %. through cooperative perception using the present system.

8.2 Outlook

As the proposed system constitutes a proof-of-concept implementation rather than aiming to be a production-ready software solution, certain crucial features were considered out of scope and several simplifying assumptions were taken to keep the focus. For instance, all aspects related to security, authentication, data integrity and validation were disregarded. Future research, development and optimization effort might complement the present work at different levels, including the following.

- **Model:** As mentioned earlier, the current model, presented in section 5.1, was designed to be easily extensible. However, it can not yet be considered complete, so the specification of an exhaustive model, that covers all potentially relevant aspects, is still required. Moreover, the current model only supports two-dimensional environments for the sake of simplicity. Therefore, it would need to be extended in order to support vertically unambiguous scene descriptions, e.g. as it is the case with a highway bridge over a rural road.

- **Timing:** Section 5.4.7 and section 7.1.2 already insinuated that timing and synchronization are crucial aspects in a CP system, though they were not thoroughly covered in this work. To maintain data consistency and avoid system failures, the problem of imperfectly accurate clocks must be addressed with more advanced techniques, e.g. using designated hardware devices [RKD11] or appropriate algorithms [JU05].

- **Fusion:** The proposed fusion algorithm is very fundamental and minimalist and comes with certain limitations. They were briefly discussed in section 5.4. More advanced techniques, like track-to-track fusion [RKD11], the combination with extrapolation and prediction of missing or incomplete observations and the integration with, for instance, Markov chain- or Bayesian network models would be desirable in future work. Similarly, a more elaborate, perhaps adaptive way of temporal decay might be added. Minor optimizations, like the use of Dubin curves for calculating spatial distance as an alternative to plain Eucledian distance, can be thought of in addition.

- **Communication:** A core point of the concept presented in this thesis is the reliance on a client-server architecture with central fusion nodes. However, a lot of research is going on about device-to-device 5G networks, which imply to revert back to VANET-like communication topologies again. Such might be considered as a serious alternative and are interesting to be put into direct comparison with the present approach.

- **Scalability:** The evaluation conducted in section 7.1 pointed out that the current system's scalability is not entirely sufficient, yet. To some extent this is due to its proof-of-concept character. However, there is still much room for improvements even beyond that. A respective set of possible measures was already presented in section 7.1.3. It includes (1) algorithmic optimizations, e.g. transitioning to a tensor representation of observations [PAKZ18] to enable them for being processed with GPU acceleration, (2) adaptive parameter optimization and (3) a more sophisticated way of publishing observations, e.g. using relevance estimation [BM13] and caching.

- **Evaluation:** Not only the system itself, but also its evaluation might be further improved in the context of future work. In addition to using a simulator, results gained in real-world tests with real 5G networks and actual, realistic traffic scenes are desirable. Moreover, an in-depth comparison of the present approach with alternative system – e.g. such based on low-level fusion or using DSRC – would be of great interest.

In conclusion, the proposed system is a modern end-to-end approach to cooperative perception, involving an elaborate software architecture and a range of novel concepts and techniques. It comes with a proof-of-concept implementation and a modular integration with a state-of-the-art autonomous driving simulator. Experiments showed its suitability to improve individual automated vehicles' perception quality and hence its potential to facilitate performance, reliability and security of autonomous driving in general. With the help of certain suggested optimizations it might find its way to become a core part of connected autonomous cars in the future. Most likely, cooperative perception solutions, like the one developed here, will play a crucial role during the early market-introduction phase of highly automated cars, which will find themselves having to operate among heavily mixed traffic.

Bibliography

[3GP19] 3GPP: 3GPP Release 15 V1.0.0. Valbonne, France, 2019. – Forschungsbericht

[5G 16] 5G Automotive Association: The Case for Cellular V2X for Safety and Cooperative Driving. Version: 2016. `https://5gaa.org/wp-content/uploads/2017/10/5GAA-whitepaper-23-Nov-2016.pdf`. 2016. – Forschungsbericht

[5G 18] 5G Automotive Association: V2X Technology Benchmark Testing. Version: 2018. `https://www.qualcomm.com/media/documents/files/5gaa-v2x-technology-benchmark-testing-dsrc-and-c-v2x.pdf`. 2018. – Forschungsbericht

[5G 19a] 5G Automotive Association: C-V2X Transforming road safety. 2019. – Forschungsbericht. – 1 S.

[5G 19b] 5G Automotive Association: Cellular V2X Conclusions based on Evaluation of Available Architectural Options. Munich, Germany, 2019. – Forschungsbericht

[AzPA$^+$19] Abou-zeid, Hatem ; Pervez, Farhan ; Adinoyi, Abdulkareem ; Aljlayl, Mohammed ; Yanikomeroglu, Halim: Cellular V2X Transmission for Connected and Autonomous Vehicles: Standardization, Applications, and Enabling Technologies. In: *IEEE Consumer Electronics Magazine* (2019)

[BM13] Breu, Jakob ; Menth, Michael: Relevance estimation of cooperative awareness messages in VANETs. In: *2013 IEEE 5th International Symposium on Wireless Vehicular Communications (WiVeC)*, IEEE, jun 2013. – ISBN 978–1–4673–6339–6, 1–5

[BMW19] BMW Group: *BMW Group erhöht die Verkehrssicherheit durch das Teilen von anonymisierten Verkehrsdaten.* `https://www.press.bmwgroup.com/deutschland/article/detail/T0296690DE/bmw-group-erhoeht-d` ie-verkehrssicherheit-durch-das-teilen-von-anonymisierten-verkehrsdaten. Version: 2019

[Bri19] Briegleb, Volker: *V2X: Telekom und BMW gegen EU-Vorschrift für vernetztes Fahren — heise online.* https://www.heise.de/newsticker/meldung/V2X-Telekom-und-BMW-gegen-EU-Vorschrift-fuer-vernetztes-Fahren-4400118.html. Version: 2019

[BSKH19] Bischoff, Maximilian ; Scheuermann, Johannes ; Kiesl, Christoph ; Hatzky, Julian: *The Edge is Near: An Introduction to Edge Computing! - inovex-Blog.* https://www.inovex.de/blog/edge-computing-introduction/. Version: 2019

[Car] Carla Contributors: *Getting started - CARLA Simulator.* https://carla.readthedocs.io/en/latest/getting_started/

[CBW+16] Chen, Qi ; Bellows, Brendan ; Wittie, Mike P. ; Patterson, Stacy ; Yang, Qing: MOVESET: MOdular VEhicle SEnsor Technology. In: *2016 IEEE Vehicular Networking Conference (VNC)*, IEEE, dec 2016. – ISBN 978–1–5090–5197–7, 1–4

[CCS18] CCS Insight: Market Forecast: 5G Connections. 2018. – Forschungsbericht

[CD93] Crowley, James L. ; Demazeau, Yves: Principles and techniques for sensor data fusion. In: *Signal Processing* 32 (1993), may, Nr. 1-2, 5–27. http://dx.doi.org/10.1016/0165-1684(93)90034-8. – DOI 10.1016/0165–1684(93)90034–8. – ISSN 0165–1684

[CM17] Calvo, Jose Angel L. ; Mathar, Rudolf: A multi-level cooperative perception scheme for autonomous vehicles. In: *2017 15th International Conference on ITS Telecommunications (ITST)*, IEEE, may 2017. – ISBN 978–1–5090–5275–2, 1–5

[CML+18] Codevilla, Felipe ; Müller, Matthias ; López, Antonio ; Koltun, Vladlen ; Dosovitskiy, Alexey: End-to-end Driving via Conditional Imitation Learning. In: *International Conference on Robotics and Automation (ICRA)*, 2018

[Col19] Collison, Ginger: *Publish-Subscribe - NATS Docs.* https://docs.nats.io/nats-concepts/pubsub. Version: 2019

[CTYF19] Chen, Qi ; Tang, Sihai ; Yang, Qing ; Fu, Song: Cooper: Cooperative Perception for Connected Autonomous Vehicles based on 3D Point Clouds. (2019), may. `http://arxiv.org/abs/1905.05265`

[Deu19] Deutsche Welle: *5G auction in Germany raises €6.5 billion from four telcoms*. `https://www.dw.com/en/5g-auction-in-germany-raises-65-billion-from-four-telcoms/a-49168657`. Version: 2019

[DG11] Dölger, Rainer ; Geissler, Torsten: *DATEX II – The standard for ITS on European Roads*. 2011

[DRC$^+$17] Dosovitskiy, Alexey ; Ros, German ; Codevilla, Felipe ; Lopez, Antonio ; Koltun, Vladlen: CARLA: An Open Urban Driving Simulator. In: *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, S. 1–16

[Elm02] Elmenreich, Wilfried: An Introduction to Sensor Fusion. (2002)

[Eri13] Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy S.: *Head First Design Patterns*. 2013. `http://dx.doi.org/10.1093/carcin/bgt051`. `http://dx.doi.org/10.1093/carcin/bgt051`. – ISBN 0596007124

[Eur] European Telecommunications Standards Institute (ETSI): *ETSI - Mobile Technologies - 5G*. `https://www.etsi.org/technologies/5g?jjj=1575453573596`

[Eur11] European Telecommunications Standards Institute (ETSI): *ETSI TS 102 637-2*. `https://www.etsi.org/deliver/etsi_ts/102600_102699/10263702/01.02.01_60/ts_10263702v010201p.pdf`. Version: 2011

[Eur19] European Telecommunications Standards Institute (ETSI): *ETSI TR 103 562*. Sophia Antipolis, 2019

[FH19] Friedman, Lex ; Hotz, George: *George Hotz: Comma.ai, OpenPilot, and Autonomous Vehicles — Artificial Intelligence (AI) Podcast*. `https://www.youtube.com/watch?v=iwcYp-XT7UI`. Version: 2019

[For08] Forbes: *Durchschnittsgeschwindigkeit in europäischen Städten — Statista*. `https://de.statista.com/statistik/daten/studie/37200/umfrage/durchschnittsgeschwindigkeit-in-den-15-groessten-staedten-der-welt-2009/`. Version: 2008

[Fro18] FROST & SULIVAN CONSULTING: Global Autonomous Driving Market Outlook, 2018 / Frost & Sulivan Consulting. Version: 2018. `https://info.microsoft.com/rs/157-GQE-382/images/K24A-2018Frost%26Sullivan-GlobalAutonomousDrivingOutlook.pdf`. Mountain View, CA, 2018. – Forschungsbericht

[GTW15] GUNTHER, Hendrik-Jörn ; TRAUER, Oliver ; WOLF, Lars: The potential of collective perception in vehicular ad-hoc networks. In: *2015 14th International Conference on ITS Telecommunications (ITST)*, IEEE, dec 2015. – ISBN 978–1–4673–9382–9, 1–5

[HKS+19] HOHM, Andree ; KLEJNOWSKI, Lukas ; SKIBINSKI, Sebastian ; BENGLER, Klaus ; BERGER, Stefan ; VETTER, Johannes ; KRUG, Sebastian: Ko-HAF – Cooperative Highly Automated Driving. 2019. – Forschungsbericht. – 270 S.

[Isr19] ISREAL HOMELAND SECURITY: *V2X Technology Successfully Tested on Ambulances*. `https://i-hls.com/archives/88787`. Version: 2019

[JU05] JULIER, S.J. ; UHLMANN, J.K.: Fusion of time delayed measurements with uncertain time delays. In: *Proceedings of the 2005, American Control Conference, 2005.*, IEEE, 2005. – ISBN 0–7803–9098–9, 4028–4033

[Kav19] KAVANAGH, Sacha: *How fast is 5G - 5G speeds and performance*. `https://5g.co.uk/guides/how-fast-is-5g/`. Version: 2019

[KBSZ14] KOHLHAAS, Ralf ; BITTNER, Thomas ; SCHAMM, Thomas ; ZOLLNER, J. M.: Semantic state space for high-level maneuver planning in structured traffic scenes. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, IEEE, oct 2014. – ISBN 978–1–4799–6078–1, 1060–1065

[KCQ+13] KIM, Seong-Woo ; CHONG, Zhuang J. ; QIN, Baoxing ; SHEN, Xiaotong ; CHENG, Zhuoqi ; LIU, Wei ; ANG, Marcelo H.: Cooperative perception for autonomous vehicle control on the road: Motivation and experimental results. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* IEEE, 2013, S. 5059–5066

[Kle18] KLEIN, Lawrence: *ITS Sensors and Architectures for Traffic Management and Connected Vehicles, by Lawrence A. Klein, published by CRC Press (a division of Taylor & Francis) in 2018.* 2018. – ISBN 13:978–1–138–74737–1

[Kor19a] Korosec, Kirsten: *GM Cruise raises $1.15B at a $19B valuation from SoftBank and Honda — TechCrunch.* https://techcrunch.com/2019/05/07/gm-cruise-raises-1-5b-at-a-19b-valuation-from-softbank-and-honda/. Version: 2019

[Kor19b] Korosec, Kirsten: *VW invests $2.6 billion in self-driving startup Argo AI as part of Ford alliance — TechCrunch.* https://techcrunch.com/2019/07/12/vw-invests-2-6-billion-in-self-driving-startup-argo-ai-as-part-of-ford-alliance/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=5E0bceEEm5cJudSF-rbcJg. Version: 2019

[LKC+13] Liu, Wei ; Kim, Seong-Woo ; Chong, Zhuang J. ; Shen, XT ; Ang, Marcelo H.: Motion planning using cooperative perception on urban road. In: *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)* IEEE, 2013, S. 130–137

[Mar17a] Markwalter, Brian: The Path to Driverless Cars [CTA Insights]. In: *IEEE Consumer Electronics Magazine* 6 (2017), apr, Nr. 2, 125–126. http://dx.doi.org/10.1109/MCE.2016.2640625. – DOI 10.1109/MCE.2016.2640625

[Mar17b] Martin, Robert C.: *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* 2017. http://dx.doi.org/10.1177/1356389011400889. http://dx.doi.org/10.1177/1356389011400889. – ISBN 978–0134494166

[McK19] McKinsey Center for Future Mobility: *Autonomous Driving — MCFM — McKinsey.* https://www.mckinsey.com/features/mckinsey-center-for-future-mobility/overview/autonomous-driving. Version: 2019

[Mey16] Meyer, M.D.: *Transportation Planning Handbook.* Wiley, 2016 https://books.google.de/books?id=MKipDAAAQBAJ. – ISBN 9781118762400

[MMKH11] Mangel, Thomas ; Michl, Matthias ; Klemp, Oliver ; Hartenstein, Hannes: Real-World Measurements of Non-Line-Of-Sight Reception Quality for 5.9GHz IEEE 802.11p at Intersections. Version: 2011. http://dx.doi.org/10.1007/978-3-642-19786-4_17. Springer, Berlin, Heidelberg, 2011. – DOI 10.1007/978–3–642–19786–4_17, 189–202

[Müt19] MÜTSCH, Ferdinand: *Basic benchmarks of 5 different MQTT brokers.* `https://muetsch.io/basic-benchmarks-of-5-different-mqtt-brokers.html`. Version: 2019

[NMTG16] NICKEL, Maximilian ; MURPHY, Kevin ; TRESP, Volker ; GABRILOVICH, Evgeniy: A review of relational machine learning for knowledge graphs. In: *Proceedings of the IEEE* 104 (2016), jan, Nr. 1, 11–33. `http://dx.doi.org/10.1109/JPROC.2015.2483592`. – DOI 10.1109/JPROC.2015.2483592. – ISSN 00189219

[Nov17] NOVAK, John: *raytriangle-test.* `https://github.com/johnnovak/raytriangle-test`, 2017

[OMGF+10] OLAVERRI-MONREAL, Cristina ; GOMES, Pedro ; FERNANDES, Ricardo ; VIEIRA, Fausto ; FERREIRA, Michel: The See-Through System: A VANET-enabled assistant for overtaking maneuvers. In: *2010 IEEE Intelligent Vehicles Symposium*, IEEE, jun 2010. – ISBN 978–1–4244–7866–8, 123–128

[Ope18] OPENSTREETMAP WIKI: *QuadTiles.* `https://wiki.openstreetmap.org/w/index.php?title=QuadTiles&oldid=1696307`. Version: 2018

[PAKZ18] PETRICH, Dominik ; AZARFAR, Darius ; KUHNT, Florian ; ZOLLNER, J. M.: The Fingerprint of a Traffic Situation: A Semantic Relationship Tensor for Situation Description and Awareness. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, nov 2018. – ISBN 978–1–7281–0321–1, 429–435

[PCY+16] PADEN, Brian ; CAP, Michal ; YONG, Sze Z. ; YERSHOV, Dmitry ; FRAZZOLI, Emilio: A survey of motion planning and control techniques for self-driving urban vehicles. In: *IEEE Transactions on intelligent vehicles* 1 (2016), Nr. 1, S. 33–55

[Pie13] PIERINGER, Christian: Modellierung des Fahrzeugumfelds mit Occupancy Grids. (2013), S. 329

[Qua17] QUALCOMM TECHNOLOGIES INC.: The Path to 5G. Version: 2017. `https://www.qualcomm.com/media/documents/files/accelerating-c-v2x-commercialization.pdf`. 2017. – Forschungsbericht

[Qua18] QUALCOMM TECHNOLOGIES INC.: C-V2X Trial in Japan. Version: 2018. `https://www.qualcomm.com/media/documents/files/c-v2x-trial-in-japan.pdf`. Tokyo, Japan, 2018. – Forschungsbericht

[RKD11] Rauch, Andreas ; Klanner, Felix ; Dietmayer, Klaus: Analysis of V2X communication parameters for the development of a fusion architecture for cooperative perception systems. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, jun 2011. – ISBN 978–1–4577–0890–9, 685–690

[RKRD12] Rauch, Andreas ; Klanner, Felix ; Rasshofer, Ralph ; Dietmayer, Klaus: Car2X-based perception in a high-level fusion architecture for cooperative perception systems. In: *2012 IEEE Intelligent Vehicles Symposium*, IEEE, jun 2012. – ISBN 978–1–4673–2118–1, 270–275

[Sch18] Schwartz, Joe: *Bing Maps Tile System - Bing Maps — Microsoft Docs*. `https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system`. Version: 2018

[SDLK17] Shah, Shital ; Dey, Debadeepta ; Lovett, Chris ; Kapoor, Ashish: AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In: *Field and Service Robotics*, 2017

[SS14] Schoettle, Brandon ; Sivak, Michael: A survey of public opinion about autonomous and self-driving vehicles in the US, the UK, and Australia / University of Michigan, Ann Arbor, Transportation Research Institute. 2014. – Forschungsbericht

[Sto93] Storey, Vede C.: Understanding semantic relationships. In: *The VLDB Journal* (1993). `http://dx.doi.org/10.1007/BF01263048`. – DOI 10.1007/BF01263048. – ISSN 10668888

[SZ12] Stiller, Christoph ; Ziegler, Julius: 3D perception and planning for self-driving and cooperative automobiles. In: *International Multi-Conference on Systems, Sygnals & Devices*, IEEE, mar 2012. – ISBN 978–1–4673–1591–3, 1–7

[Tam18] Tampa Hillsborough Expressway Authority: *THEA Connected Vehicle Pilot - Program Overview 2018 November*. `https://www.youtube.com/watch?v=wz4WvdGD1Bg`. Version: 2018

[TSG19] Thandavarayan, Gokulnath ; Sepulcre, Miguel ; Gozalvez, Javier: Generation of Cooperative Perception Messages for Connected and Automated Vehicles. (2019), aug. `https://arxiv.org/abs/1908.11151`

[Ver14] Verkehrslenkung Berlin (VLB): Verkehrsstärkenkarte PKW / Verkehrslenkung Berlin (VLB). Version: 2014. `https://www.berlin.d` e/senuvk/verkehr/lenkung/vlb/de/erhebungen.shtml. Berlin, 2014. – Forschungsbericht

[WCHW12] Wood, Stephen P. ; Chang, Jesse ; Healy, Thomas ; Wood, John: The potential regulatory challenges of increasingly autonomous motor vehicles. In: *Santa Clara L. Rev.* 52 (2012), S. 1423

[WDT+13] Wang, Yunpeng ; Duan, Xuting ; Tian, Daxin ; Lu, Guangquan ; Yu, Haiyang: Throughput and Delay Limits of 802.11p and its Influence on Highway Capacity. In: *Procedia - Social and Behavioral Sciences* 96 (2013), nov, 2096–2104. http://dx.doi.org/10.1016/J.SBSPRO.2013.08.236. – DOI 10.1016/J.SBSPRO.2013.08.236

[Wik19a] Wikipedia: *5G.* https://en.wikipedia.org/w/index.php?title=5G&oldid=929191112. Version: 2019

[Wik19b] Wikipedia: *Datex II.* https://en.wikipedia.org/w/index.php?title=Datex_II&oldid=919193268. Version: 2019

[Wik19c] Wikipedia: *Negation as failure.* https://en.wikipedia.org/w/index.php?title=Negation_as_failure&oldid=878307515. Version: 2019

[Wik19d] Wikipedia: *Sicherheitsabstand.* https://de.wikipedia.org/w/index.php?title=Sicherheitsabstand&oldid=193799907. Version: 2019

[Wik19e] Wikipedia: *Vehicle-to-grid.* https://en.wikipedia.org/w/index.php?title=Vehicle-to-grid&oldid=916597472. Version: 2019

[WKW+18] Wolf, Peter ; Kurzer, Karl ; Wingert, Tobias ; Kuhnt, Florian ; Zollner, J. M.: Adaptive Behavior Generation for Autonomous Driving using Deep Reinforcement Learning with Compact Semantic States. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, jun 2018. – ISBN 978–1–5386–4452–2, 993–1000

[WL17] Wevers, Kees ; Lu, Meng: *V2X Communication for ITS - from IEEE 802.11p Towards 5G.* https://futurenetworks.ieee.org/tech-focus/march-2017/v2x-communication-for-its. Version: 2017

List of Tables

List of Figures

Appendix A

Supplementary Texts

A.1 Background

A.1.1 Levels of Autonomy

- **Level 0 ("Active driver"):** No computer assistance of any kind. A car is completely controlled by its human driver.
- **Level 1 ("Feet off"):** Basic assistance, e.g. adaptive cruise control. While most functions are controlled by the driver, the car might take responsibility of a single task, e.g. accelerating and decelerating in certain scenarios.
- **Level 2 ("Hands off"):** Partial automation, e.g. cruise control and lane centering. At this level, a car is able to take over multiple driving tasks in combination. While the driver is still required to monitor the roadway, she is "disengaged from physically operating the vehicle" [Kle18] and may keep her hands of the steering wheel and feet of the pedals. To ensure that a driver still pays full attention and is able to intervene in case of system failures or critical situations, various methods of *Driver Monitoring* are employed. Such include to visually observe a driver's face using cameras or to measure the force applied to the steering wheel.
- **Level 3 ("Eyes off"):** High degree of automation. At this level, a driver might fully rely on a car's self-driving under most conditions, delegating all safety-critical function to the ADAS. Usually, it would maintain a comprehensive awareness of its environment and is able to react on it. Although a driver still has to be present and prepared to take occasional control, she is not required to constantly monitor the traffic.
- **Level 4 ("Attention off"):** Full automation. This refers to a system that is able

to “perform all safety-critical driving functions and monitor roadway conditions for an entire trip.“ [Mey16] At this level, there is no necessity for a driver to actually occupy the vehicle.

- **Level 5 (“Passive passenger“):** Full autonomy. The highest level of automation describes a system that is capable of driving under any conditions, even extreme ones. Its performance is expected to be at least human-like or even surpass human driving capabilities.

With this classification, it is worth noting that only the highest level actually refers to the term “autonomy“. [WCHW12] states that although this term is in more widespread public use, speaking of “automation“ would be more accurate for levels 1 to 4. Only Level 5 cars are self-governing and may take independent decisions, e.g. selecting a destination and an appropriate route, while cars of all other levels still have a human person in the driver’s seat.

A.2 Related Work

A.2.1 Further Modeling and Representation Approaches

Another standard exists with **DATEX II**, specified by the European Committee for Standardization [DG11]. The XML-based format is meant for *“exchanging traffic information between traffic management centres, traffic service providers, traffic operators and media partners“* [Wik19b], however, not particularly for cooperative perception. It defines a way to describe traffic events, such as road works or congestions as well as information on the current parking situation. However, it is not suitable to describe particular traffic situations in high detail. The same holds true for the **SENSORIS** representation format presented by [HKS$^+$19], that was designed in a different way, but with the same purpose.

[SZ12] follows yet a different approach as the authors show a way to represent the current local world state as the instantiation of a **Markov Logic Network**, in which weights represent uncertainty about the true state of an observation. This representation is inherently graphical and by incorporating first-order logic, the underlying model also allows for basic inference, in theory. A specification of what objects and relations to include to comprehensively model a traffic scene is not provided.

A.3 Concept & Design

A.3.1 Geographical Sharding Schema

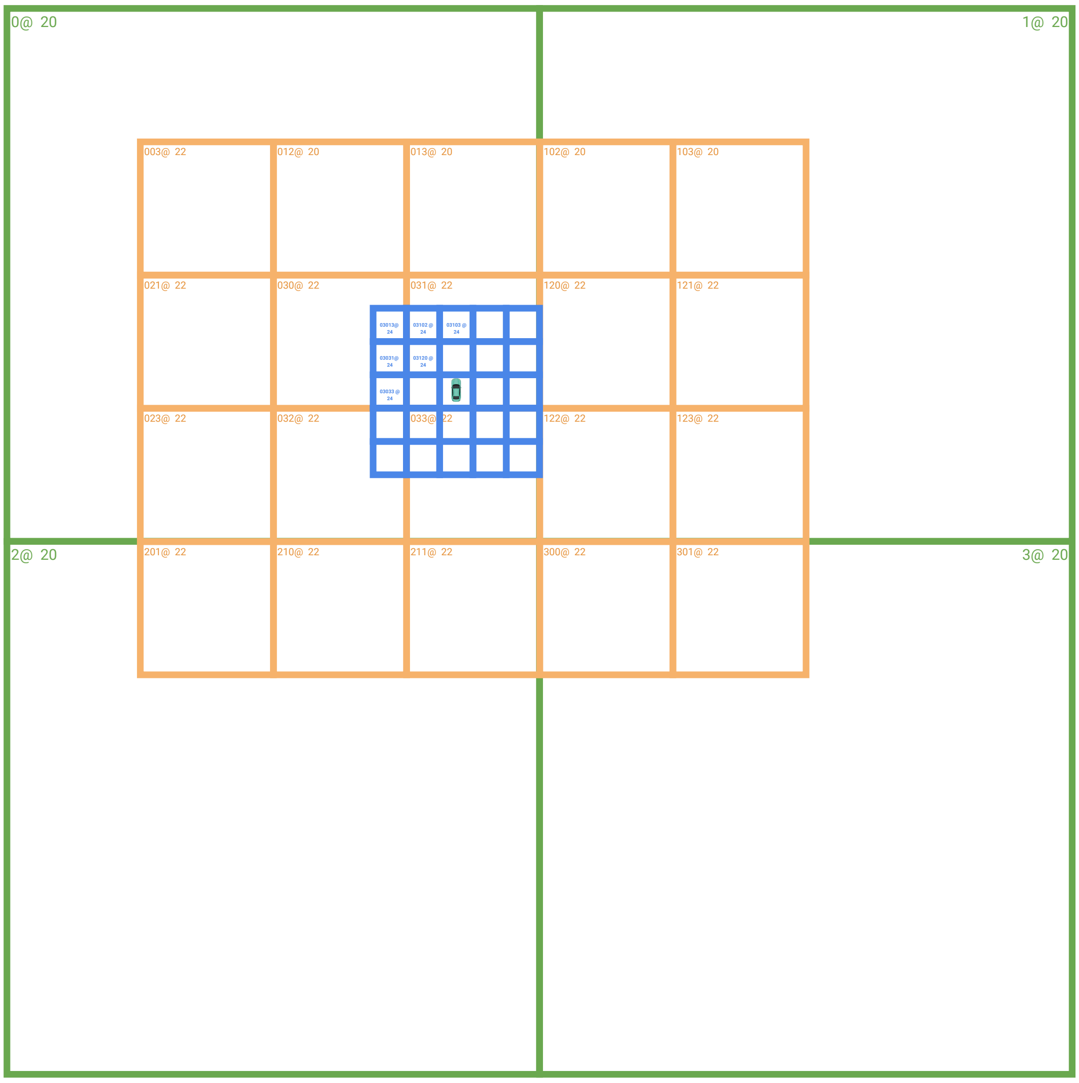


Figure A.1: Schematic Illustration of Geographical Sharding
Blue cells (= type 1 tiles) are part of the vehicle's observed occupancy grid.
Orange cells (= type 2 tiles) are part of the vehicle's range of interest.
Green cells (=type 3 tiles) are subject to geo distribution.

A.4 Evaluation

A.4.1 Perception Evaluation Analyses

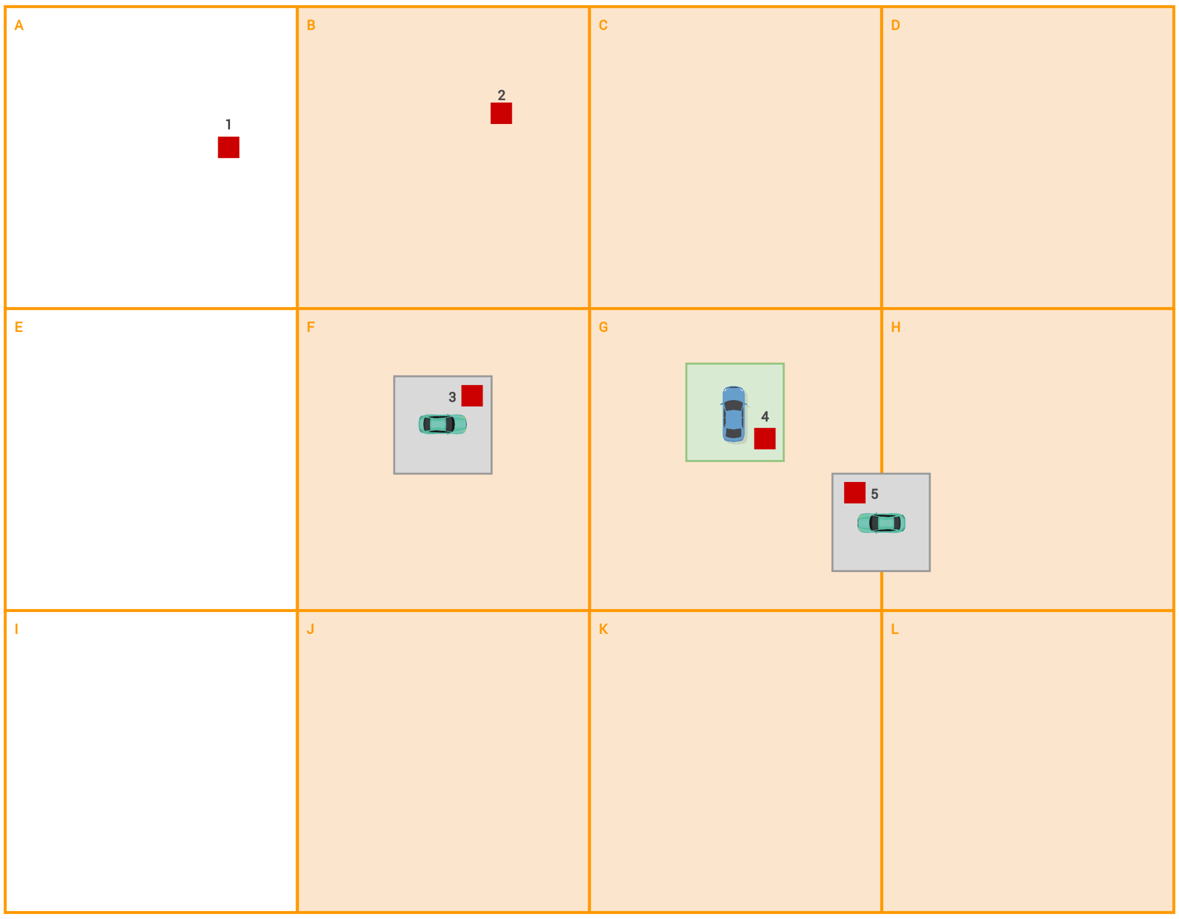


Figure A.2: Schematic Illustration of Evaluation-Relevant Cells

The blue vehicle is the ego vehicle in this example, while the green vehicles are other network participants. The orange type 2 cells are those, which the ego is currently subscribed to (i.e. its "neighborhood") for receiving observations from the fusion node. At the same time, they are the ones whose type 1 cells to include into the evaluation. Red rectangles depict obstacles and green or gray rectangles around vehicles show their respective observation range, i.e. local grid size.

Obstacle 4 is within the ego's local observation range, while 2, 3 and 5 can only be recognized through CP with the help of other cars. Obstacle 1 is out of range for the ego, regardless of CP being turned on or off.

Without CP, the ego can detect $\frac{1}{4}$ obstacles in potential range, while with CP it virtually detects $\frac{3}{4}$ of them.

Appendix B

Source Code

B.1 SQL Queries for Traffic Volume Estimation

B.1.1 Query: Geographic Area

```
/* Get area of specified bounding box im km^2 */
/* Output: 21.252402298715576 */
SELECT st_area(
    st_transform(
      st_makeenvelope(13.40666, 52.519444, 13.447532, 52.493904, 4326),
      3857
    )
  ) / (1000 * 1000) AS area;
```

B.1.2 Query: Total Road Length

```
/* Get total street length in km within given bounding box */
/* Output: 159.31409120764036 */
SELECT
  sum(st_length(way)) / 1000 AS total_length
FROM planet_osm_line
WHERE
  way && st_transform(
    st_makeenvelope(13.40666, 52.519444, 13.447532, 52.493904, 4326),
    3857
  )
  AND highway IN ('primary', 'secondary', 'tertiary', 'residential');
```

B.1.3 Query: Average Number of Lanes

```sql
/* Get average number of lanes per way within given bounding box */
/* Output: 2.4029850746268657 */
SELECT
  avg(to_number(lanes, '99'))
FROM planet_osm_line
WHERE
  way && st_transform(
    st_makeenvelope(13.40666, 52.519444, 13.447532, 52.493904, 4326),
    3857
  )
  AND highway IN ('primary', 'secondary', 'tertiary', 'residential');
```

B.2 Ray Casting Intersection Algorithm

```
// ray_intersect.cu
// Source: https://gamedev.stackexchange.com/a/103714/130059
__device__ float rayBoxIntersect ( float3 rpos, float3 rdir, float3 vmin, float3 vmax )
{
   float t[10];
   t[1] = (vmin.x - rpos.x)/rdir.x;
   t[2] = (vmax.x - rpos.x)/rdir.x;
   t[3] = (vmin.y - rpos.y)/rdir.y;
   t[4] = (vmax.y - rpos.y)/rdir.y;
   t[5] = (vmin.z - rpos.z)/rdir.z;
   t[6] = (vmax.z - rpos.z)/rdir.z;
   t[7] = fmax(fmax(fmin(t[1], t[2]), fmin(t[3], t[4])), fmin(t[5], t[6]));
   t[8] = fmin(fmin(fmax(t[1], t[2]), fmax(t[3], t[4])), fmax(t[5], t[6]));
   t[9] = (t[8] < 0 || t[7] > t[8]) ? NOHIT : t[7];
   return t[9];
}
```

Appendix C

Evaluation Results

C.1 Serialization Benchmark

```
$ go test -bench=.

goos: linux
goarch: amd64
BenchmarkGob-12             5172            222607 ns/op
BenchmarkCapnp-12           4129            317576 ns/op
BenchmarkProto-12           7460            169135 ns/op
PASS
ok      _/home/ferdinand/dev/talkycars-thesis/src/evaluation/serialization
CreateGob:       0.5322 ms/msg,         8.9372 KB/msg
CreateCapnp:     0.5907 ms/msg,         15.8170 KB/msg
CreateProto:     0.4396 ms/msg,         8.3755 KB/msg
```

C.2 MQTT Broker Benchmark

```
# mqtt-bench 0.3.0 amd64-linux
# mosquitto 1.6.8 amd64-linux

# $COMMAND="mqtt-bench -action p -broker tcp://192.168.179.40:1883 -count 10000 -qos 1 \
    -clients $CLIENTS -size $SIZE"

# $SIZE=46000
# $CLIENTS=1 eval $COMMAND
Result : broker=tcp://192.168.179.40:1883, clients=1, totalCount=10000,
    duration=2202ms, throughput=4541.33messages/sec

# $CLIENTS=10 eval $COMMAND
Result : broker=tcp://192.168.179.40:1883, clients=10, totalCount=100000,
    duration=17319ms, throughput=5774.01messages/sec

# $CLIENTS=100 eval $COMMAND
Result : broker=tcp://192.168.179.40:1883, clients=100, totalCount=1000000,
    duration=239050ms, throughput=4183.23messages/sec

# $SIZE=12000
# $CLIENTS=1 eval $COMMAND
Result : broker=tcp://192.168.179.40:1883, clients=1, totalCount=10000,
    duration=1277ms, throughput=7830.85messages/sec

# $CLIENTS=10 eval $COMMAND
Result : broker=tcp://192.168.179.40:1883, clients=10, totalCount=100000,
    duration=10275ms, throughput=9732.36messages/sec

# $CLIENTS=100 eval $COMMAND
Result : broker=tcp://192.168.179.40:1883, clients=100, totalCount=1000000,
    duration=114590ms, throughput=8726.76messages/sec
```