D5.2

Storage of Data from Smart Vehicle's Internal Network

Topic	SU-ICT-01-2018
Project Title	Artificial Intelligence-based Cybersecurity for Connected and Automated Vehicles
Project Number	833611
Project Acronym	CARAMEL
Contractual Delivery Date	M18
Actual Delivery Date	M18
Contributing WP	WP5
Project Start Date	01/10/2019
Project Duration	30 Months
Dissemination Level	Public
Editor	UCY
Contributors	PANA, FICOSA, GREENFLUX, i2CAT, 0INF

Document History		
Version	Date	Modifications
0.1	15/06/2020	Initial document structure, table of contents
0.5	30/12/2020	50% content
1.0	01/03/2020	First draft for internal review
1.5	15/03/2020	Ready for consortium review
2.0	20/03/2020	Ready for Submission to SAB
3.0	31/03/2020	Submitted to EC

Contributors		
Name	Organisation	Contribution
Christos Kyrkou	UCY	Editor(s)
Christos Laoudias	UCY	Content
Ilias Theodorou	UCY	Content
Arnab Barua	UCY	Content
Josep Escrig	i2CAT	Content
Petros Kapsalas	PANASONIC	Content
Bob Elders	GREENFLUX	Content
Anish Khadka	0INF	Content

DISCLAIMER OF WARRANTIES

This document has been prepared by CARAMEL project partners as an account of work carried out within the framework of the contract no 833611.

Neither Project Coordinator, nor any signatory party of CARAMEL Project Consortium Agreement, nor any person acting on behalf of any of them:

- makes any warranty or representation whatsoever, express or implied,
    - with respect to the use of any information, apparatus, method, process, or similar item disclosed in this document, including merchantability and fitness for a particular purpose, or
    - that such use does not infringe on or interfere with privately owned rights, including any party's intellectual property, or
- that this document is suitable to any particular user's circumstance; or
- assumes responsibility for any damages or other liability whatsoever (including any consequential damages, even if Project Coordinator or any representative of a signatory party of the CARAMEL Project Consortium Agreement, has been advised of the possibility of such damages) resulting from your selection or use of this document or any information, apparatus, method, process, or similar item disclosed in this document.

CARAMEL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 833611. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

DISCLOSURE STATEMENT

"The following document has been reviewed by the CARAMEL External Security Advisory Board as well as the Ethics and Data Management Committee of the project. Hereby, it is confirmed that it does not contain any sensitive security, ethical or data privacy issues."

Table of Contents

List of Figures

List of Tables

List of Acronyms

CAVs	Connected and Autonomous Vehicles
ADAS	Advanced Driver-Assistance System
API	Application Programming Interface
APS	Auto Parking System
AVI	Audio Video Interleave
CAN	Controller Area Network
CDF	Cumulative Distribution Function
CDR	Charge Details Record
CLAHE	Contrast Limited Adaptive Histogram Equalisation
CP	Control Pilot
CPO	Charge Point Operators
DGPS	Differential Global Positioning System
DL	Deep Learning
DSO	Distribution System Operators
ECU	Electronic Control Units
eMSP	Electro Mobility (Service) Provider
EV	Electric Vehicle
GHE	Global Histogram Equalisation
GPS	Global Positioning System
GSOP	Guaranteed Standard of Performance
IMU	Inertial Measurement Unit
LCE	Live Capture Engine
LHE	Local Histogram Equalisation
ML	Machine Learning
MVs	Meter Values
OCPP	Open Charge Point Protocol
OSCP	Open Smart Charging Protocol
RGB	Red Green Blue
RFID	Radio-frequency identification
USB	Universal Serial Bus
V2X	Vehicle-to-everything

Executive Summary

Connected and Autonomous Vehicles (CAVs) are expected to eliminate the major limitations and shortcomings of existing transportation, such as safety, accessibility, productivity, efficiency, and environmental pollution. However, recent studies showed that advanced driver-assistance and autonomy systems can be compromised by spoofing attacks, adversarial machine learning techniques to scene structural elements such as traffic signs, and other malicious acts. Due to awareness of the security gaps and weaknesses of autonomous vehicles, both the academia and industry have conducted research in novel techniques for anomaly detection and robustification methods. These enhance the overall security and safety of the vehicles, especially in terms of confidentiality, integrity, and authenticity. Primarily, these techniques rely on collecting huge amounts of data to train machine learning algorithms.

Hence, a major objective within the CARAMEL project is to facilitate the data collection process from the various components of the CAV ecosystem. However, raw data derived from the vehicle's internal network and other infrastructure can potentially have errors or be invalid. Therefore, a data collection and pre-processing pipeline should be developed, to overcome such problems in order to prepare the data for use by the anti-hacking device that will be used for connected and autonomous vehicles. To achieve this, the data used by the various detection and mitigation systems are first described and appropriate pre-processing techniques are implemented for each case. The main goal of such systems is to detect anomalies, without affecting the normal operation of the autonomous vehicles. A significant benefit is the ability to detect environmental attacks and unknown anomalies, by adopting ML and Deep Learning (DL) techniques. Furthermore, the development of this component will facilitate the real-time dashboard and database-driven analytics and the visualisation of the current state of the vehicle internal network.

1 Introduction

To enable advanced features such as safety and increased autonomy in Advanced Driver-Assistance Systems, Machine Learning (ML) techniques are often applied. Such techniques rely on training on generated and collected data to be able to perform tasks such as detecting anomalies, that may affect the normal operation of the Connected and Autonomous Vehicles (CAVs). To organize and summarize the huge amounts of generated data the necessary components for collection and storage of data from the various vehicle networks (CAN, MOST, LIN, FlexRay, Ethernet, etc.) and pre-processing techniques used by the various detection and mitigation systems, need to be developed. The raw data derived from the vehicle's internal network can potentially have errors or be invalid. Hence, a key feature of this component will be to process them through a series of operations like data cleaning, data aggregation, and data transformation. Through this procedure redundancy will be removed and also data will be structured according to the appropriate format that will be needed for further processing in order to prepare them for use within the anti-hacking device.

1.1. Purpose of this Document

This deliverable will provide specifications and implementation details of the CARAMEL collection and data storage solution from the various components such as the autonomous vehicle, infrastructure, as well as simulation environments. Emphasis is given on data preparation capabilities that are essential to create good machine learning systems. In particular techniques relevant to pre-processing of the data, aggregation and filtering, removal of outliers, and normalization will be discussed. As shown in Figure 1, the collection and storage of data will allow a user to access both synthetic and real data and apply the necessary pre-processing steps for a given machine learning algorithm thus enabling rapid experimentation. All these procedures will further assist in the design, development and prototype implementation of CARAMEL antihacking device.

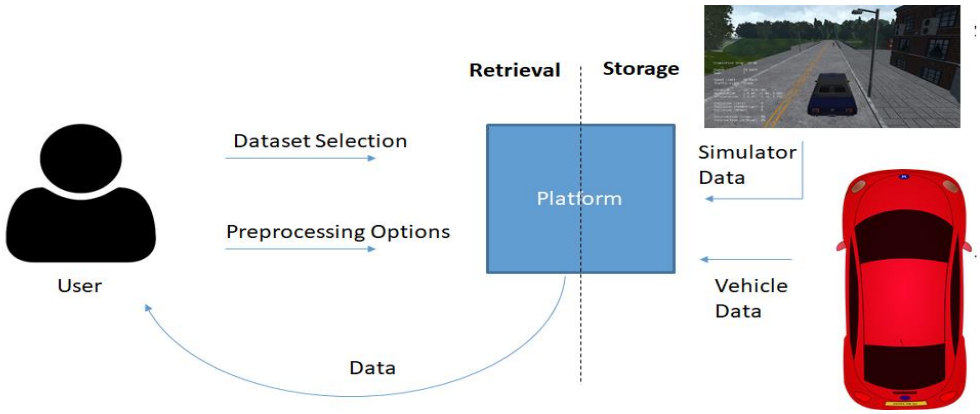


Figure 1: Example concept of the storage platform and data sources for the autonomous vehicle case.

1.2. Structure of this Document

The remainder of this document is organized as follows:

• **Section 2** – Description of the data storage platform for the autonomous vehicle data, with details on the architecture, access protocols and front-end information as well as description of the data stored during the vehicle's operation. Details are also provided for the different simulators and synthetic data used to complement the real data.

• **Section 3** – Description of the data collection methodology from the various sensors of the autonomous vehicle and the infrastructure of the electric charging stations. We also provide a description of some potential use-case scenarios for testing an autonomous vehicle.

• **Section 4** – Provides a description of the data and various pre-processing steps that are necessary to prepare them for use in the anti-hacking device or for training a machine learning algorithm. The description focusses on camera sensor data, GPS data, V2X message data, and electric charging station data.

• **Section 5** – Concludes this document.

1.3. Relation to other Tasks and Deliverables

D5.2 derived in Task 5.1, is related to the following tasks and deliverables:

- **Task 2.1 - Use Cases Elaboration / D2.1 Report on Detailed Specification of Use Cases**: D5.2 receives the definition of the CARAMEL components and the use cases, as well as the technical evaluation strategy deployed within the project.

- **Task 2.3 - Analysis of Security and Privacy Requirements / D2.3 Specifications of CARAMEL Security and Privacy Requirements**: D5.2 considers the user, security and privacy requirements defined in D2.2 and D2.3, with focus on the definition of the various scenarios for each CARAMEL use case.

- **Task 2.4 - System Specifications and Architecture / D2.4 - System Specifications and Architecture**: D5.2 receives the system architecture from D2.4, including the definition of attack types, interfaces and the requirements.

- **Task 3.2 - Cyberthreat Detection Using Sparse and Deep Priors / D3.2 Cyberthreat Detection Using Sparse and Deep Priors (interim)**: D5.2 receives the requirements for data preprocessing from D3.2.

- **Task 3.3 - Cyberthreat Detection and Response Techniques for Cooperative Automated Vehicles / D3.4 Cyberthreat Detection and Response Techniques for Cooperative Automated Vehicles**: D5.2 receives the requirements for data preprocessing from D3.4.

- **Task 3.4 - Task 3.4 Cyberthreat Detection and Response Techniques for Plug-in Electrical Vehicles/ D3.5: Cyberthreat Detection and Response Techniques for Plug-in Electrical Vehicles**: D5.2 receives the requirements for data preprocessing from D3.5.

- **Task 4.2 - AI-based Context-rich and Context-aware Cybersecurity/ D4.2 Robust to Cyberattack Machine Vision Models Based on Improved Training Methods and Anomaly Detection Deep Networks**: D5.2 receives the data specifications and preprocessing requirements from D4.2.

- **Task 4.1 - PKI-enabled Vehicle Identity Management Against Identity Theft / D4.1 CARAMEL PKI-enabled Vehicle Identity Management System:** D5.2 receives the requirements for data preprocessing from D4.1.

- **Task 4.3 - Holistic Situational Awareness with ML Application / D4.3 CARAMEL Situational Awareness Solution based on the Machine Learning Applications**: D5.2 receives the data specifications and preprocessing requirements from D4.3.

- **Task 5.2 Advanced Algorithmic Detection of Attacks via passive Anti-hacking Device / D5.3 Machine Learning based Detection of Attacks into Anti-hacking Device**: D5.3 will integrate the preprocessing steps and data formatting derived from D5.2 in the anti-hacking device.

2 *Data Storage Platform*

2.1 Autonomous Vehicle Data

This section presents information related to the Panasonic computing and data storage system integrated into the demo vehicle. CAN bus and Flexray bus enable access to vehicles data and can be used to transfer data between different nodes. The Panasonic system integrated into the demo vehicle, has a Flexray network to transfer vehicle data between different parts inside the vehicle and between the vehicle and Panasonic system. Also, it has a local CAN network to transfer data between Panasonic components, but this bus is not connected to the demo vehicle.

To transfer data the Panasonic software (SW) framework shall interact with the demo vehicle inputs and outputs which consists of:

- Main Flexray network : Ego vehicle data (i.e. speed, steering, wheels pulses, blinkers,….).
- Local CAN network : used for local communication between Panasonic system devices (i.e. Micro Autobox, CarPC, …).
- Sonars Sensors ECU CAN network : used for receiving data from the sonars sensors.
- Cameras : 4 cameras mounted in the car (front, rear, left, right)

In order for the Panasonic SW to be able to communicate with those interfaces , an interface module is needed. The Live Capture Engine (LCE) is the main module handling data inputs/outputs between Panasonic SW and the vehicle internal data and sensors.

2.1.1 Architecture

The diagram in Figure 2 describes the system level architecture, as well as with the connection between devices and components such as:

- CarPC : the main device in which Panasonic algorithm run.
- Micro Autobox : the controller , control the vehicle actuators.
- Vector CAN/Flexray cases : Vector VN series devices are needed to connect to CAN/Flexray buses , it is connected to the D-Sub connector of the buses and provide a USB interface to access the data on the buses.
- Camera splitter and converter: Camera frames data is being transmitted over LVDS connection so a splitter is needed to output duplicated LVDS then a convertor is needed to convert from LVDS to USB connection.
- Emergency button : in case of any non-controlled behaviour , emergency button is pressed to disconnect power from all devices and cut the access to all CAN/Flexray buses

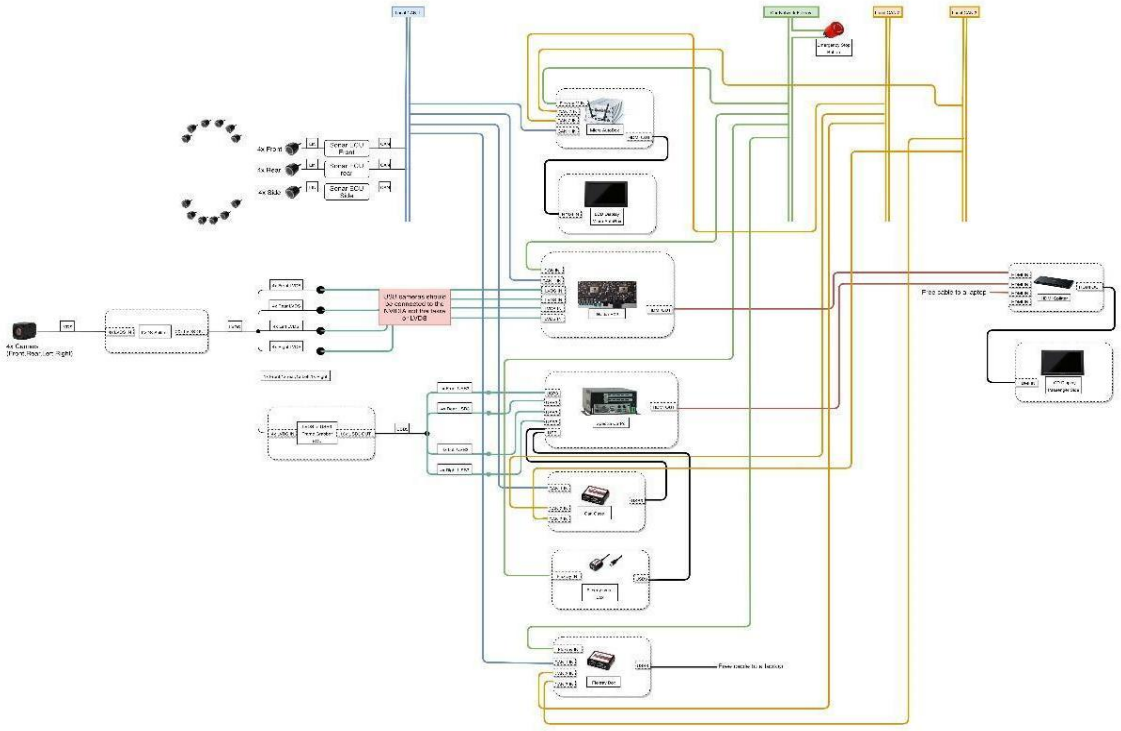


Figure 2: System level architecture, and connection between devices and components

2.1.2 Access and Front-End

LCE is the main component to interact with vehicle inputs and outputs. The following interfaces are used :

- CAN/Flexray interface: using vector libraries API's , LCE can access data (send and receive)on network buses through Vector VN boxes.
- Cameras USB interface : using Opencv libraries API's , LCE can retrieve cameras frames.

After capturing all needed data , post processing can be done and then LCE passes data and frames to all other relevant components to run needed algorithms.

After the Panasonic framework finishes running all needed algorithms, data need to be sent to the controller (Micro Autobox) is being sent to the LCE, then LCE sends the data on the Local CAN bus to the controller. Once the Controller receives all needed data, it will start controlling the vehicle using the Flexray bus interface.

2.1.3 Storage

During the operation of the Panasonic framework, the LCE can record and store all data being processed and passed to the other SW modules including:

- Vehicle odometry
- Camera frames

The following files are being stored as show in Figure 3:

- .avi files : Stores video frames (one file per camera)
- CAN.txt : Stores odometery data and state of controller .
- descriptor.yml : Stores metadata related to the vehicle and the environment (camera calibration data , weather situation , location, …)

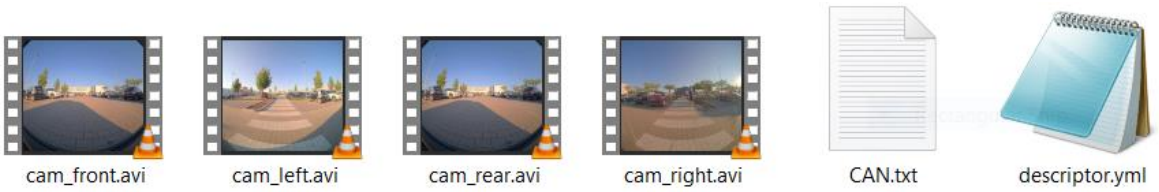


Figure 3: Files stored through the Panasonic framework

2.2 Simulated and Synthetic Data

Two simulators are used to produce synthetic data. CARLA is a driving simulator that can collect data from virtual sensors (cameras, LIDAR, GPS, etc.) installed in cars driving in realist 3D scenarios. On the other hand, SUMO is a traffic simulator that can produce realistic traces of individual vehicles driving in complex environments like congested urban areas. From these traces, it is possible to produce an extreme number of synthetic and realistic V2X messages.

2.2.1 CARLA data

To increase the diversity of the dataset across driving and environmental conditions we also employ the CARLA simulator to construct a synthetic dataset for use in some of the CARAMEL use-cases such as the ones outlined in T4.2 and T4.3. CARLA is a powerful open-source tool which supports flexible specification of virtual sensors and dynamic environmental conditions and the full control and configuration of all static and dynamic actors mainly vehicles and pedestrians. We deploy an autonomous driving controller on a vehicle agent under diverse environmental conditions such as dynamic weather and different virtual towns to generate synthetic data. This driving simulator, also provides a convenient way to train autonomous driving agents with a variety of different sensor suites (cameras, LIDAR, depth, etc.), and test them in realistic traffic scenarios.

The basic idea is that the CARLA simulator itself acts as a server and waits for a client to connect. A Python process connects to it as a client. The client sends commands to the server to control both the car and other parameters like weather, starting new episodes, etc. The server (i.e., the simulator) sends measurements and images back to the Python process. The Python client process can then print the received data, process it, write it to disk, etc. By default all the communication between the client and the server happen on TCP ports 2000, 2001 and 2002. The messages sent and received on these ports is explained here, but it is not very important to understand everything over there, as most of the client-server communication is abstracted by the carla module in the PythonClient directory.

The first step in collecting simulated data is to get images by automated driving. In addition, to sensor data we also want to get semantic segmentation ground truth to test the different attack detection algorithms with. We are saving each image (frame) to disk as a .png file as it is coming in. If the sensor is an RGB camera, it stores it as an RGB image. But if it is semantic segmentation ground truth, then it removes all but the red channel, because it is the only channel with any useful information. If the sensor type happens to be a depth camera, it converts the information in the three channels into a single “channel” of floating point data.

We run the simulator in fixed time-step mode. This means we need to use the -benchmark flag and provide an fps=<framerate> argument while starting the simulator. We use a sampling rate of 10 Hz in order to collect both sequence data as well as still frames which corresponds to a frame rate of 10. We also run the simulator in synchronous mode to make sure that the Python client is able to keep up with all the data that the simulator generates. Running in synchronous mode forces the simulator to wait for a control signal from the Python client before sending the next packet of data. Examples of generated images are shown in Figure 4.

Figure 4: Data Generated from the CARLA simulator

2.2.2 SUMO data

A large dataset of V2X messages is needed in T4.1 to train a ML model that aims to track vehicles from the V2X messages that these vehicles send. Ideally, this dataset must contain all the V2X messages that would send all the vehicles (considering that all the vehicles are connected vehicles) circulating in a large city during 24 hours. It is expected that all the vehicles manoeuvres and traffic situations will be covered with these large dataset.

The V2X messages must contain at least the following information as it is stated by ETSI[1]:

- Heading: identification ID or pseu-identification of the vehicle
- Basic information: position of the vehicle and timestamp
- Status: speed, heading, acceleration and curvature

Additionally, the V2X messages must be sent continuously by the connected vehicles. The time in between two messages sent by a connected vehicle must be from 100 to 1000 milliseconds. Since nowadays it is impossible to obtain this dataset from experimental data, it needs to be produced from simulated and synthetic data.

SUMO is an open source traffic simulator. It performs microscopic and continuous traffic simulations in urban and inter-urban environments. It can generate realistic vehicle traces of vehicles circulating in complex scenarios. Figure 5 shows some examples from the simplest to the most complex: a) is a road with two lanes in each direction and a zebra crossing, b) is a large round about with several entries and exits with one or two lanes, c) has crossroad with many lanes and several zebra crossings and traffic lights, d) is a highway with one entry and one exit in each direction, e) is small sector of a city and f) is a whole city.

[1] ETSI EN 302 637-2, “Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service”

Figure 5: SUMO traffic scenarios

A SUMO simulation that has been taken as a reference in the present project is the simulation performed by Uppoor et al. (2014)[2]. This simulation produced a dataset containing the position and velocity for each second of more than 700.000 vehicles driving in the city of Cologne (400 square kilometers) for a period of 24 hours. This extreme dataset is publicly available for non commercial purposes and it has been used as a base to generate the V2X messages dataset needed in T4.1. In section "4.2 Data used for V2X messages Tracking Attack" the descriptions of the V2X messages dataset and the data processing methods applied to produce it are detailed.

3 Data Collection Methodology and Scenarios

3.1 Description of Autonomous Vehicle Sensors

Due to the required safety of manoeuvres of autonomous vehicles, these vehicles require a high precision perception of their surrounding environment. All the possible information ranging from the details of motions of used ego vehicle to the motion of surrounding pedestrians can lead the vehicle to make an optimal decision at the right time to avoid any collision. This level of safety confidence can be just archived using an optimal fusion between the collected information of various mounted sensors on the vehicle.

In this section a short description of mounted sensors on Panasonic Automotive Systems Europe (PASEU) automated vehicle is introduced. The proposed system is a vision-based autonomous system including several components as explain below:

Cameras: four wide angle ("fish-eye") (30 frame/sec with a 190° field of view) cameras which are mounted on the side mirrors, the front, and rear bumper of the test vehicle to detect the free spaces and obstacle around the vehicle simultaneously. The detailed specifications of the used cameras are as shown in Table 1:

Table 1: Technical specifications of PASEU cameras

Parameter	**Information-Size**
Sensor Model	Sony ISX016
Image Format	Parallel Output YUV422
Serializer Model	FPD Link-Ⅲ
Effective Area	H：1296×V：976

[2] S. Uppoor, O. Trullols-Cruces, M. Fiore, J.M. Barcelo-Ordinas, **Generation and Analysis of a Large-scale Urban Vehicular Mobility Dataset**, *IEEE Transactions on Mobile Computing*, Vol.13, No.5, May 2014

Output Resolution	H：1280×V：960 (Cropping)
Frame rate	30 (Frame/S)
Data logging	54 (MB/S)

Sonars: sonar sensors (10 Hz low range up to 6 m) in front, rear, left, and right of the vehicle to collect additional data about the close object-obstacle to the vehicle.

Velodyne LiDAR Sensor: for the further validation of the performance of the vision-based parking system a 360-degree laser scanner is used. In recent modern "level five" of autonomous vehicles (e.g. Google car), LiDAR sensors are mainly used to collect the information of the surrounding area to feed the perception sections. Lidar technology is still in its infancy, and cost-effective sensors are not yet readily available on the market. In the Panasonic driving platform, scanning lidars are therefore only used for validation. The technical specification of the used sensor is as shown in Table 2:

Table 2: Technical specifications of the used laser scanner in PASEU

Parameter	**Information-Size**
Sensor Model	HDL-32E
Number of Channels	32Up to 100(m)
Range Accuracy	Up to ±2(cm)
Field of View (Vertical)	+10.67 to -30.67 (41.33) (Deg)
Angular Resolution (Vertical):	1.33 (Deg)
Field of View (Horizontal)	360 (Deg)
Angular Resolution (Horizontal/Azimuth)	0.1 – 0.4 (Deg)
Rotation Rate	5 – 20 (Hz)

Differential GPS and its internal Inertial Measurement Unit (IMU)**:** the position of the vehicle can be measure precisely at each position using an Applanix Differential GPS (DGPS). The precise position pf the vehicle with the captured information of the surrounding of the vehicle helps the autonomous system to ensure the performance of other mounted sensors due to the accurate collected ground truth. The accuracy of the used DGPS sensor is as below:

Table 3: Technical specifications of the mounted DGPS sensor on the test vehicle of PASEU

Parameter	**DGPS**	**With Post Processing**
Position (m)	0.3-0.5	0.02-0.05
Roll/Pitch (Deg)	0.015	0.015
Heading (Deg)	0.02	0.02

Vehicle network: the related information and collected data from the vehicle are transferred to each component of the system via a local CAN and FlexRay system of the vehicle. This internal information regarding each vehicle motion is updated as follows:

Table 4: FlexRay messages and their update frequencies in PASEU test vehicle.

Parameter	**Update frequency (ms)**
Steering wheel position	20
Gear position	5

Rack position	20
Wheels rotation counter	20
Wheels speed	20
Blinker information	10

Data logger: All the captured data of used sensors are stored on the onboard car-pc of the vehicle. Regarding syncing the data together, currently the live capture engine (Win7 64-bit with Intel(R) Xeon(R) CPU 3.50 (GHz) processor) which is responsible for data logging considers the time stamp of receiving the data (e.g. from each Camera) on the PC and store them accordingly. The logger does not consider the capture time stamp of the data itself.

When the data logger receives a new data, it assigns a timestamp to it from a global time stamp service that is starting from 0.00000 second and is always incrementing.

Ego vehicle: in the automotive industry ego vehicle term is mainly used to refer to the test vehicle which is manipulating the requested tasks. In our system, the whole APS system is mounted on a test vehicles: Mercedes Benz C-Class 2014 (S204).

Figure 1 introduces a simplified version of the autonomous driving system which has been used in PASEU.

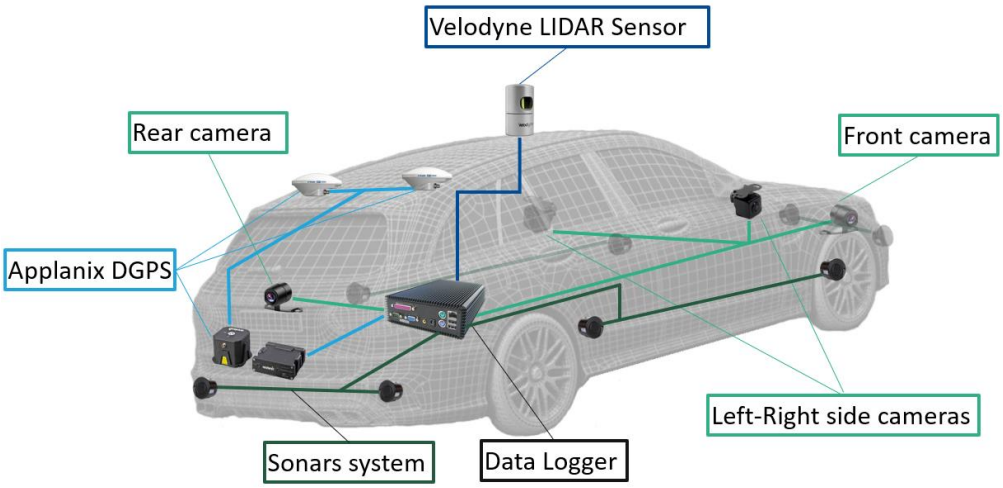


Figure 6: Panasonic autonomous driving system setup on one of the test vehicles (Mercedes Benz C-Class).

3.2 Description of Infrastructure for Electric charging stations

This section describes information flows between EV charging stations and their central systems. Part of the information exchanged is temporarily stored (up to one month) while other messages are permanently stored.

Charging Process

The interaction starts with the EV driver plugging in the charging cable and swiping his charge card to start a charging session. Figure 7 describes the messages flows between involved parties.

1. **Authentication** A customer holding a valid RFID card (i.e., authentication token) uses it to authenticate himself at the charging station and waits until its validity is checked. Authentication by the CP requires interaction with the CPO and eMSP. The CP extracts the UID of the card and sends an authentication request to the CPO. The CPO contacts the

eMSP and checks if the customer is actually authorised to charge at such a CP. A response is then generated and traced back to the CP and to the customer afterwards.

2. **Charging** In case the customer is allowed to charge, i.e., if authentication is successful, the cable is locked (a pin on the inside of the socket is automatically moved through the car plug). The CP starts the charging by creating an OCPP transaction session that will lock the socket until the customer decides to re-authenticate again. While in charging mode, the CP sends meter readings (MeterValues) to the CPO every 3 minutes. These MeterValues are stored in a database. When the customer wishes to unplug the cable, he has to re-authenticate himself again to the CP (using the RFID card). In most of the cases, this second authentication is performed locally at the CP and does not require any interaction with the CPO. After the session is completed a Charge Details Record (CDR) is forwarded to the eMSP. This CDR includes the customer UID and the total amount of energy charged, the CP identifier (where the charge took place) and the starting and ending time of the transaction.
3. **Billing** After the whole process is completed (i.e. after charging the car), the eMSP bills the customer. This is usually done on a monthly basis and according to the contracted service.

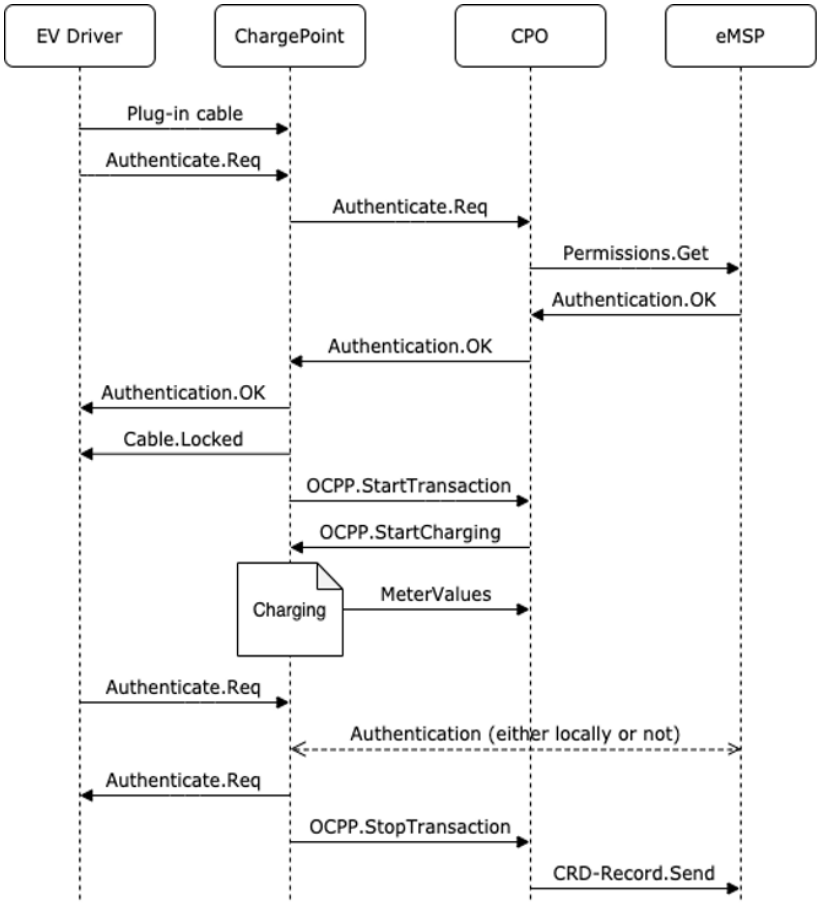


Figure 7: Message flows between parties involved

Smart Charging

To be able to forecast and divide the assigned capacities per cable and per eMSP for specific time slots of the next 24 hours, the DSO relies on some variables: historical usage data (of the DSO meter in the transformer), aggregated meter readings per CPO and per cable (from the CPO meter in the CP) and weather forecasts. The DSO makes a forecast based on the historical usage data and weather forecasts of the next 24 hours and uses the aggregated meter readings to divide the capacity per CPO and per cable. The historical usage data is obtained directly from the meter placed in the transformer serving the specific cable. Weather forecasts are retrieved through a web service designated for this purpose. Figure 8 depicts the message flows between the involved parties.

After calculating the forecasts, the DSO sends them (using the OSCP protocol) to the CPO, which in turn forwards them to the corresponding CPs (using the OCPP protocol). Upon reception of the forecasts an acknowledgement message is always sent back to the DSO. The CPs will be able to charge the cars according to the capacity dynamically assigned at each time slot.

Every 24h the CPO can then send the aggregated meter readings per cable to the DSO (the combined meter readings of all CPO meters in all CPs on a cable), such that the DSO can use those values to divide the forecasted capacity and provide them to the energy supplier for billing purposes.

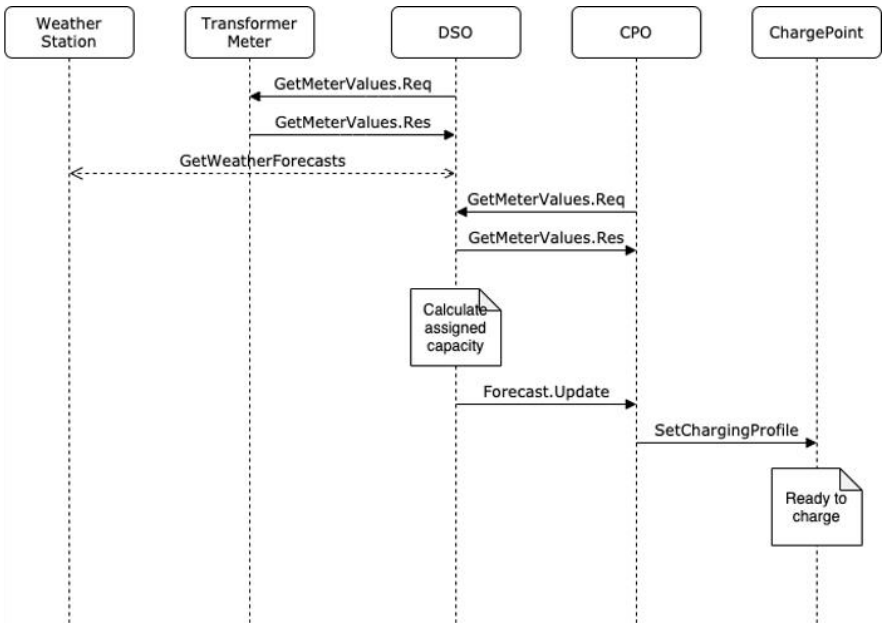


Figure 8: Smart Charging message flow

GreenFlux Database

The term smart charging is used to stress the importance of the data exchange between an electric vehicle and a charging station, as opposed to traditional charging where the charging devices aren't connected to the cloud. This connection of the charging device to the cloud allows the charging station owner (in our use case GreenFlux) to monitor, manage, and restrict the power supply of each station achieving resources' optimization and ensuring the grid from overcharging issues and a potential blackout.

The cloud-enabled solutions for smart charging allows modifications of the system's features on-the-go without the need for upgrade of the existing infrastructure, offering an effortless and secure way to add or remove features at will. A smart charging solution is a sustainable investment as any potential change on the demands can be turned into new features and added to the current infrastructure. Since the charging stations are connected to the cloud, they can be managed based on various signals (features in the database) such as: the requested volume (kWh), local electricity consumption, the start time of the charging, the requested amperage, the amount of other vehicles being charged or electrical devices being used on a nearby premise. The inclusion of heterogeneous data requires the installation of a smart charging system that can prioritize the requests from different clients and eventually create a more sustainable energy system based on renewable energy sources.

In the context of CARAMEL, the GreenFlux EV charging dataset from GreenFlux's lab in Amsterdam will be used comprised of millions of charge sessions hosted on the cloud platform from throughout the Netherlands dating back to 2012. The database consists of four tables, each one of them representing **a unique entity** in the EV scenario, namely: the Charge Points, the Charge Detail Records (CDRs), the Connections and the Meter Values (MVs).

The table Charge Points represents the actual charging points connected to the GreenFlux platform and contain information for their geo-location features, such as address, zip code and city as depicted in Table 5.

Table 5: An overview of the ChargingPoint table of the GreenFlux's database

Column	Datatype	Description
ID	PK, Int	Unique ID for Charge Points
ExternalID	Unique ID for Charge Points	External charge point identifier
Address	Nvarchar (255)	Charge point address
Zipcode	External charge point identifier	Charge point zipcode
City	Nvarchar (max)	Charge point city
Country	Charge point address	Charge point Country (NLD = Netherlands)
Latitude	Nvarchar (max)	Charge point latitude coordinate
Longitude	Charge point zipcode	Charge point longitude coordinate

The table Charge Detail Records (CDRs), as shown in Table 6, describes the necessary details of each charging attempt such as the duration and the volume, but it also includes features from other tables as foreign key in order to express the correlation with the other entities of the grid. Therefore, every record to the database includes the unique ID of the charge card used by the EV driver, and the unique ID of the charging station.

Table 6: An overview of the Charge Detail Records table of the GreenFlux's database

Column	Datatype	Description
ID	PK, int	ID for CDR

Duration	Nvarchar (50)	Duration of session
Volume	Nvarchar (50)	Volume in kWh
AuthenticationId	Nvarchar (50)	Unique charge card ID
ChargePoint_ID	FK, int	Unique Charge Point ID
ConnectorId	Nvarchar(255)	ChargePoint Connector Identifier
dStart	datetime	Session start time
dEnd	datetime	Session end time

The different tables in the GreenFlux's database represent different entities in the grid, and the linking of the entities in the database is achieved through the table Connections, as shown in Table 7, whose goal is to accomplish this connection.

Table 7: An overview of the Connections table of GreenFlux's database

Column	Datatype	Description
ID	PK, int	Connection identifier
ConnectorId	int	ChargePoint Connector Identifier
ChargePoint_ID	FK, int	Unique ChargePoint ID

The last table in the GreenFlux's database is the Meter Values table, shown in Table 8, which consists of information about the actual values of the charging process. Apart from the technical details of the table, it contains the connection_ID as a foreign key in order to be linked with the other entities of the system.

Table 8: An overview of the Meter Values Table of GreenFlux's database

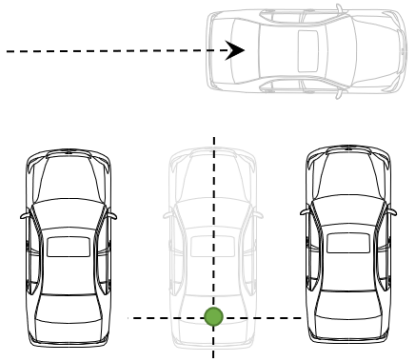
Column	Datatype	Description
ID	PK, int	Meter Value identifier
Timestamp	datetime	In-session timestamp
Value	Decimal(18,2)	Measured Value
ValueType	int	Specifies unit of measurement
ReadingContext	int	(0 = Wh, 1 = kWh, 8 = Amp)
Connection_ID	FK, int	Specifies measurement or instruction

3.3 Scene and Scenario Description

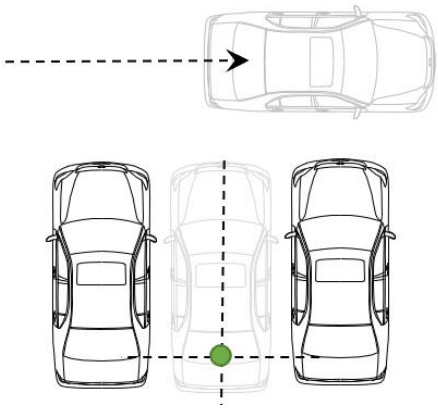
There are various ways of testing an autonomous vehicle. A use case, to be qualified to be tested for an autonomous system, shall be able to cover the requirements of the system. Here each test case of the test procedure is designed carefully to fulfil the criteria of the auto parking system. These scenarios can be the critical scenarios in which the auto parking systems can fail to detect or plan a safe maneuver to the target position. Some of critical parking scenarios which are used in Panasonic Automotive Systems are listed in Table 9:

Table 9: Some examples of the tested use cases in Panasonic Automotive Systems.

Use Case 1: Normal perpendicular parking scenario in which the parking width is larger than the car width plus 80 cm:	**Use Case 2**: Narrow perpendicular parking scenario in which the parking width is less than the car width plus 80 cm:

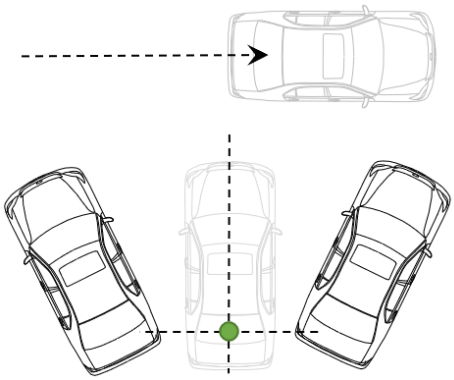


Here, the detected parking places is wide, and the ego vehicle has enough distance (more than 1 m) to the parked vehicles.



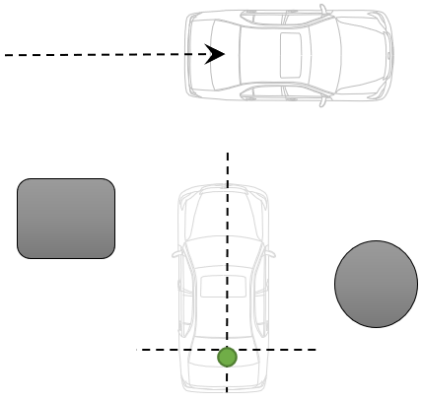
The same as normal use cases but the width of the parking place is less than vehicle width plus 80 cm.

Use Case 3: Misaligned perpendicular parking scenario:



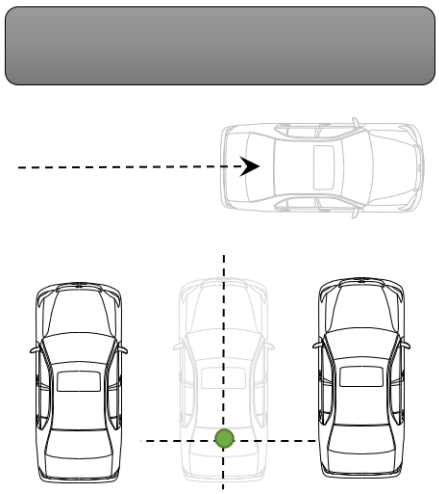
Parked vehicles besides the proposed parking place are parked in an unusual and misaligned shape.

Use Case 4: A perpendicular parking scenario between unusual objects:



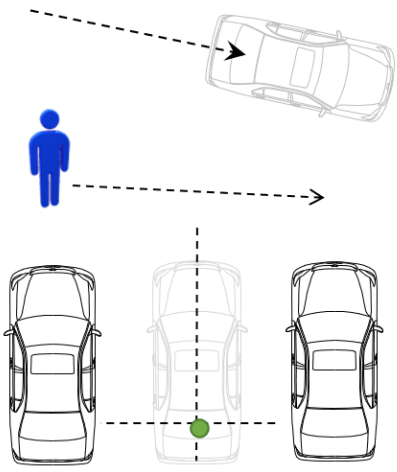
Parking between small or unusual objects (e. x. trees, parking pillars, and etc.).

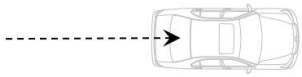
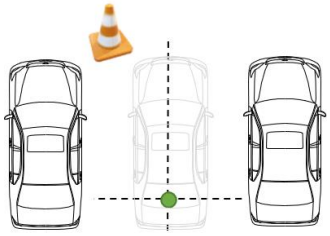
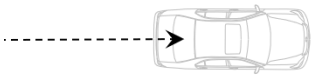
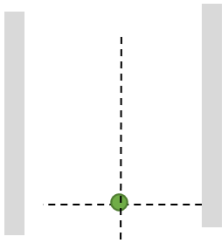
Case 5: Normal parking scenario with limited moving space:



Parking where the road width is limited.

Case 6: Parking with moving objects



	Use cases in which the ego vehicle is not parallel to the parking place and several moving objects cross the parking area.
Use Case 7: Occupied parking place with small objects such as a kid:   The parking place is occupied by small objects. It means the vehicle is not allowed to park in the parking place.	**Use Case 8**: Parking between parking lines:   Parking based on the parking marks with no parked vehicle around the proposed parking place.

To ensure the best performance of mounted sensors on the vehicle while logging data of the environment the motions of the vehicle shall satisfy the following motion criteria as listed in Table 10:

Table 10: Motion criteria of vehicle while logging data

Parameter	Accepted range
Vehicle speed	0-20 (kph)
Yaw rate	0-45 deg/sec

By defining the scenarios, the next step is driving the ego vehicle through the scenarios and collecting the data from the scenes using the on-board mentioned sensors. It can be done in several ways. In Panasonic Automotive Systems the data logging is done in two main steps as follow:

A. Simulated Test Cases

The synthetic scenarios can be used to test the algorithms of the autonomous driving technology for very critical situations in which the ego vehicle needs to be pushed to its limits. In real world, these scenarios are difficult to test due to the high risk of collision with other vehicles or even to moving pedestrians. Apart from the mentioned safety factor, the simulated scenarios can be set as Software in the Loop (SIL) test to run and test the application for long term evaluations as well.

The simulated scenarios are produced and tested as explained in Figure 9.

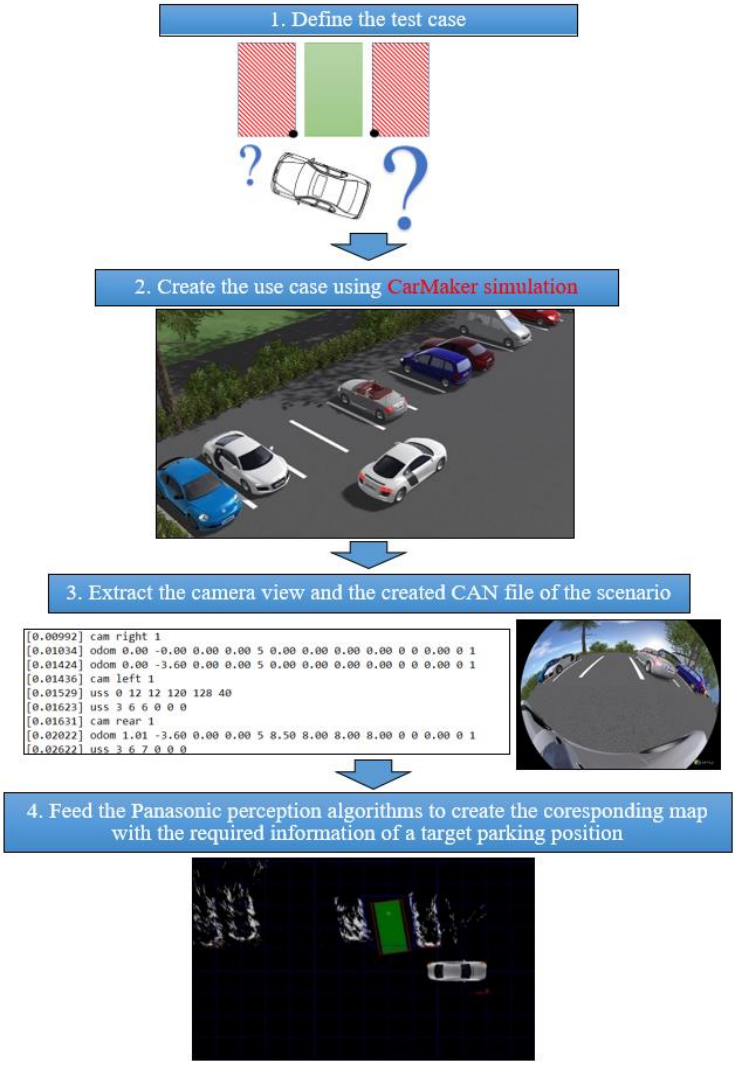


Figure 9: Data logging, test and validation process for the simulated scenarios

The above-mentioned steps are processed to create an offline simulated use case for the mentioned scenarios in Table 9 as some of the most critical parking scenarios. Each use case is fully defined by its positions and dimensions as explained precisely and the required ground truth of the vehicle motion can be easily extracted from the simulation environment. The collected extracted ground truth of the created scenario can help the algorithms to be tested and evaluated precisely.

By having the required information of each simulated test case, the scenario is simulated in the simulation environment of the IPG CarMaker software and the related data of the scenario are extracted (such as camera information and CAN data of the test vehicle) to feed the perception part of the parking system.

Figure 10: An example of an identical test case, created in IPG CarMaker (Right-side) environment and the comparison between a recording from the real world (Left-Side).

Using the collected logged data of the created scenarios, the perception part of the parking system uses the data and creates the corresponding static map of the environment including all the relevant data regarding the detected objects (see Figure 10). Knowing the information of the parking area a free parking place can be proposed to the auto parking system.

By changing the dimensions of the test cases, the whole test can be repeated several times and the results of the tests are stored for further evaluations. In this case, a test case is tested for all the possible changes in the requested parking scenarios (e.g. vehicle speed). Clearly, the simulated scenarios cannot cover all the actual scenarios of real parking scenarios in the real world due to the limitations of the simulation software, but they can be used to test the algorithm and its performance in a safe and cheap manner. The advantages and disadvantages of this method have been listed in Table 11:

Table 11: The advantages and disadvantages of synthetic test scenarios

Advantages	Disadvantage
Cheap to run	**Not fully realistic**
Can be tested for several use cases continuously	**Artificial rendered inputs which cannot fully show the performance of the system in the real world**
Fast and easy to change the scenario	**The dynamic of the vehicle cannot be fully simulated**
Does not need a lot of resources	**The possible errors and noises in real world cannot be fully simulated and implemented**
Can be applied to very dangerous and critical scenarios	**-**

B. Data Logging from Real World Information

By having the results of the test and evaluation of the application for the simulated scenarios, as the next step, the application is tested with the data provided by the real world as well. Because it will be unsafe to directly test the vehicle in real world in real-time, autonomous driving applications are extensively evaluated over several recorded scenarios in offline mode as well.

The data logging for offline scenarios at Panasonic Automotive Systems are done using the following process as illustrated in Figure 11.

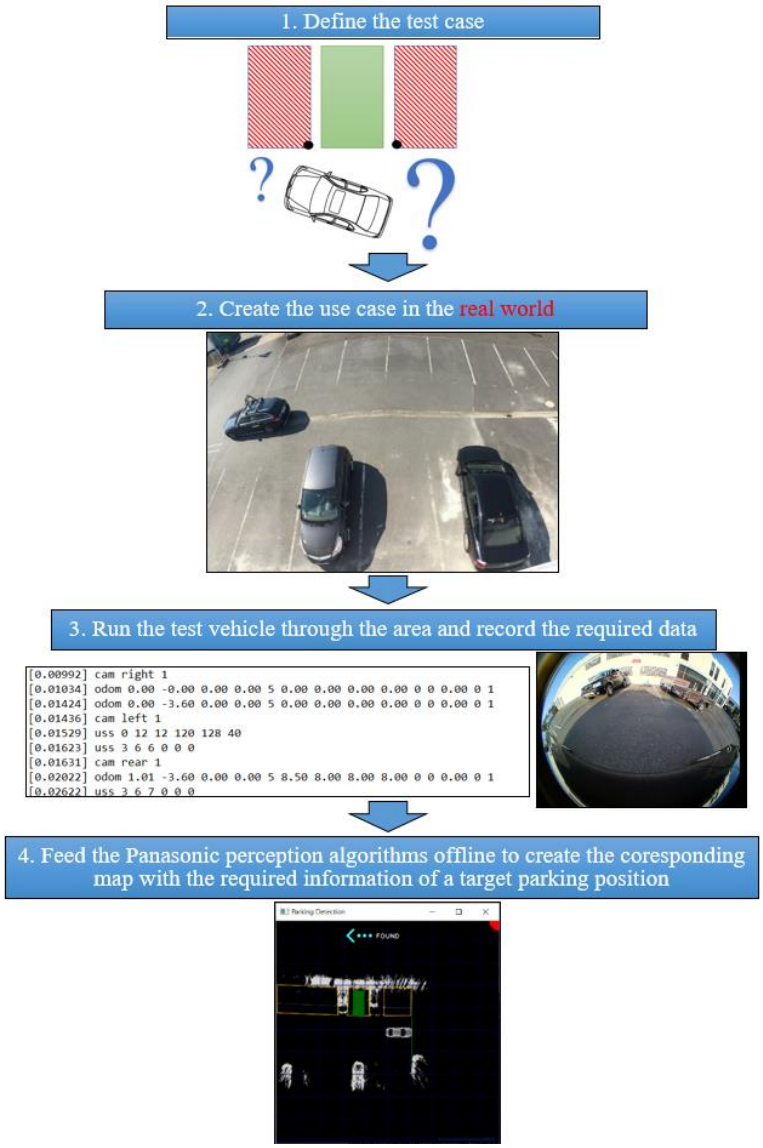


Figure 11: Data logging, test and validation process for the real-world scenarios

As in the simulated test cases, the offline test procedure is started by defining the required scenario. The test cases and their required inputs such as the dimensions of the proposed parking places, position of the side objects, and so on are defined first. Then, by having the requirements of the test case, the actual situation is created in a real-world parking area. The test vehicle which has a recording software on it, is driven manually trough the parking area and the required data such as the camera images from the side cameras, CAN data, and other provided information are recorded from the provided scenarios.

The recorded data are tested on a test bench and if are qualified, are fed to the auto parking system (APS) in an offline mode. In this case, the relative map of the area is created offline and the information of the proposed parking places are provided for the trajectory planner (Figure 12).

Figure 12: An example of a test case, in offline mode (using a test laptop) with the real world data.

The same as the previous section, the auto parking system receives the desired parking position from the perception part and plan the moving manoeuvres accordingly.

By changing the dimensions and positions of the parking objects, the test scenario is redesigned and the whole procedure is repeated. This offline test is useful to test the performance of the algorithm for the real scenarios in a safe and controllable manner. Table 12 listed the advantage and disadvantage of this type of test process:

Table 12: The advantages and disadvantages of offline test scenarios.

Advantages	**Disadvantage**
Safe to test all the scenarios	Very time consuming
Test with real data of the real world	Tested with high performance devices which can effect on the performance of the algorithm
Realistic results of the performance of the system	Very difficult to re-arrange a test case

The algorithm and its performance are tested carefully for several test cases (more than 1500 scenarios) including simulated and offline test cases and there were several steps in which the planner was edited, modified and improved due to the results of the tests.

4 Data Description and Format

Data is at the heart of any machine-learning-based solution. Within CARAMEL different solutions will utilize different types of data depending on the scenario and targeted sensor. In addition, since CARAMEL targets very selected and limited scenarios it is a challenge to collect sufficient volume of real-world data. In such cases hybrid data are also employed that combine collected real data and generated synthetic data from realistic simulators. Herein we describe data that will be primarily used for the evaluation of the different scenarios, but that can also be used for training machine learning models.

4.1 Data used for Camera Sensor Attacks

4.1.1 Carla Synthetic Data

The RGB camera within CARLA generates realistic pictures of the scene (Figure 13). In addition, the semantic segmentation camera renders elements in scene with a different color depending on how these have been tagged (Figure 13). The attributes available for this camera can be made to match the RGB camera. These are summarized in Table 13.

Table 13: Data Collected from Carla simulator

RGB	Provides clear vision of the surroundings. Looks like a normal photo of the scene.
Semantic segmentation	Renders elements in the field of view with a specific color according to their tags.

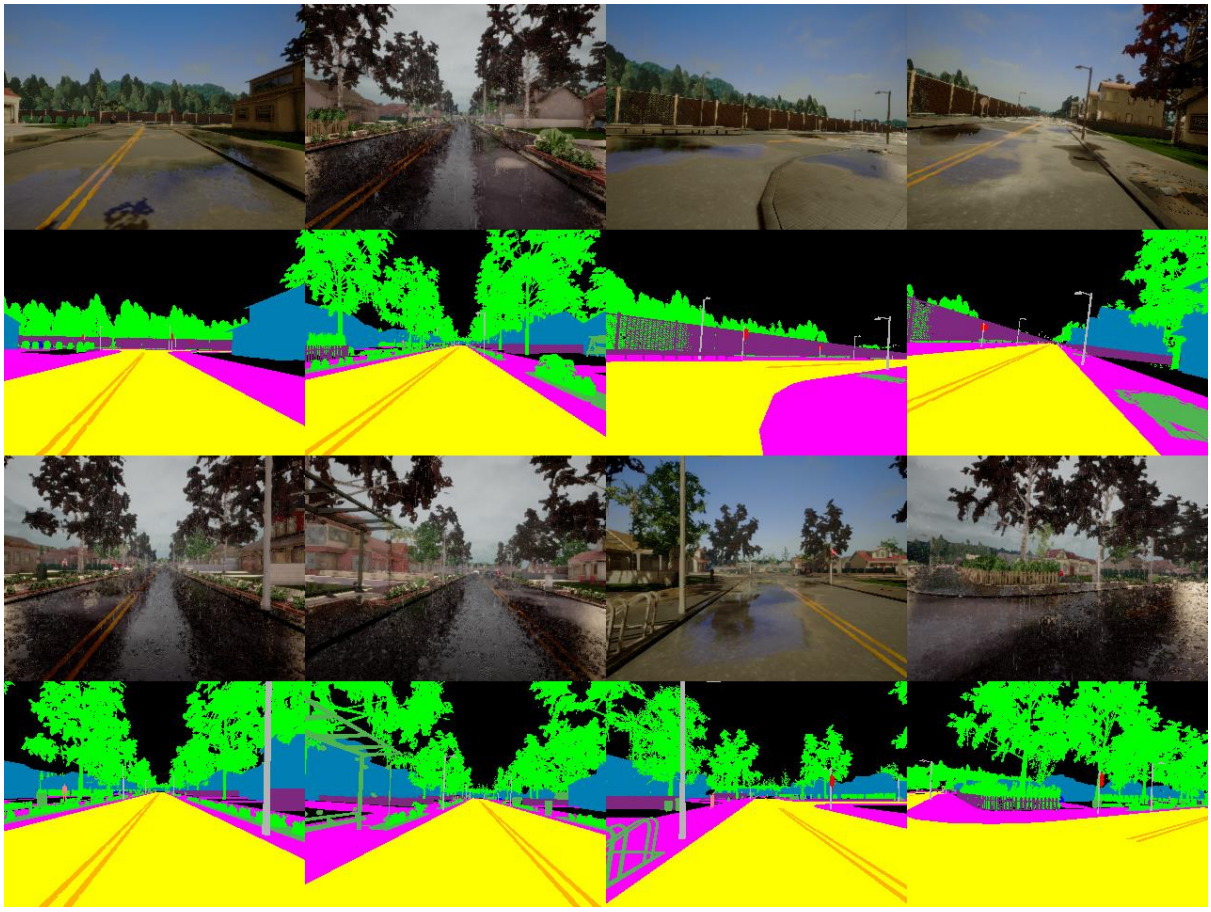


Figure 13: Example of semantic segmentation data captured from the CARLA simulator

4.1.2 Data Processing

Herein we describe the image processing pipeline through which the image data will be made readily available for the machine learning solution in the anti-hacking device. We build the image pre-processing pipelines through the widely used Open CV library [1] as shown in Figure 14.

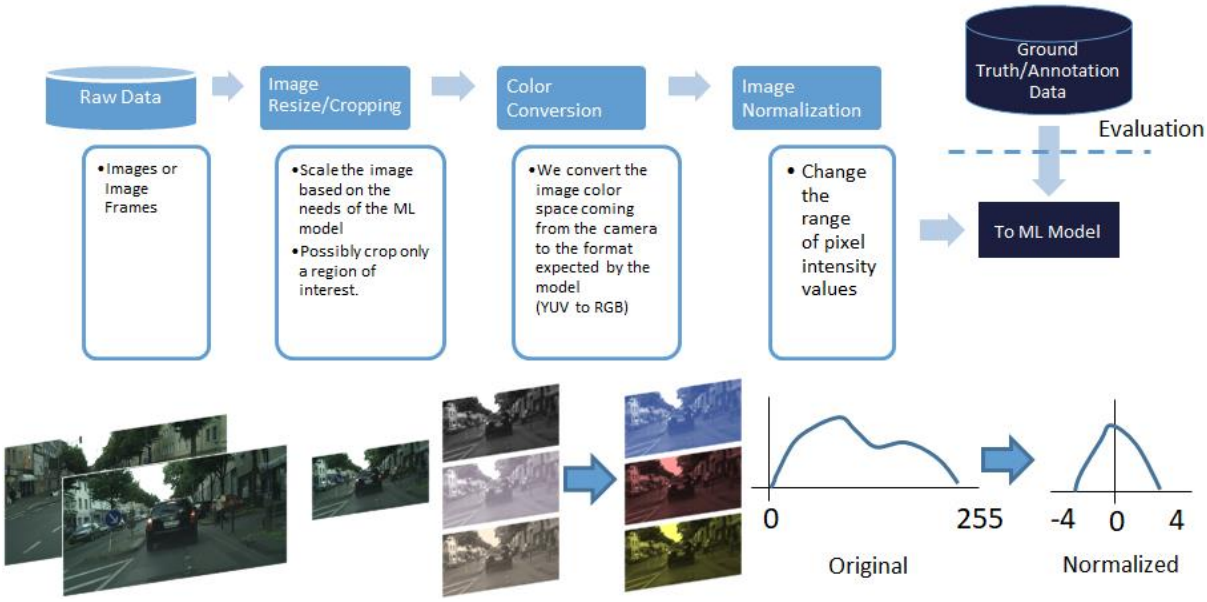


Figure 14: Image Processing Pipeline for the ML-based denoising solution

4.1.3 Image Resizing

Image resizing is necessary to reduce its spatial dimensions to those that the model expects. Additionally, it is important to resize the image to reduce the computational requirements. Resizing an image means changing the dimensions of it, be it width alone, height alone or both. Also, the aspect ratio of the original image could be preserved in the resized image. Bilinear interpolation is a widely used method for image resizing that works by interpolating pixel color values, introducing a continuous transition into the output even where the original material has discrete transitions. Although this is desirable for continuous-tone images, this algorithm reduces contrast (sharp edges). Bicubic interpolation yields substantially better results, with an increase in computational cost.

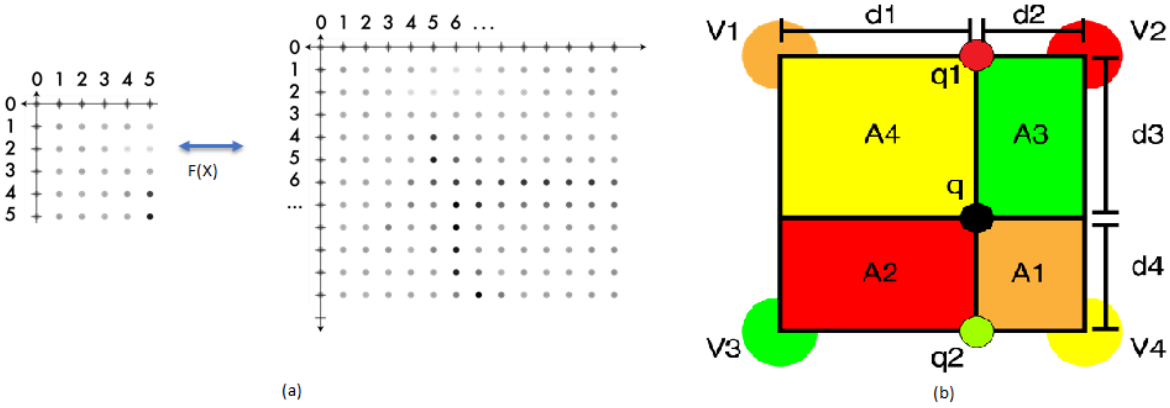


Figure 15: (a)The resizing functions changes the spatial resolution of the image either to reduce it or increase it. (b) How bilinear interpolation with pixel area works.

We resize an image,through OpenCV using its **cv2.resize()** function which allows us to change the spatial dimensions of the as shown in Figure 15-a. We choose the bilinear interpolation due to its

improved speed over the bicubic. Bilinear interpolation is performed using linear interpolation first in one direction, and then again in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location. A variant of bilinear interpolation is used for resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. An illustration of this is shown in Figure 15-b. The value of a pixel q can be retrieved by using Eq. 1.

$$q = V_1 \times A_1 + V_2 \times A_2 + V_3 \times A_3 + V_4 \times A_4$$
$$where\ A_1 = d_2 \times d_4,\ A_2 = d_1 \times d_4,\ A_3 = d_2 \times d_3, A_4 = d_1 \times d_3 \quad \textbf{(Eq. 1)}$$

The $A_i$correspond to the area of each of the 4 neighboring pixel values given by the distances d_i to pixel q that we are looking for. If we choose a coordinate system in which the four points where f is known are (0, 0), (1, 0), (0, 1), and (1, 1), then the interpolation formula can be further simplified.

4.1.4 Image Color Conversion

Color space conversion is the translation of the representation of a color from one basis to another. This typically occurs in the context of converting an image that is represented in one color space to another color space. In practical applications, camera modules may capture images in specific formats to encode an image or video in a way closer to the human perception [2]. We can convert the image colorspace,through OpenCV using its cv2.cvtColor() function. Some popular colorspaces include RGB, HSV, and YUV shown in Figure 16.

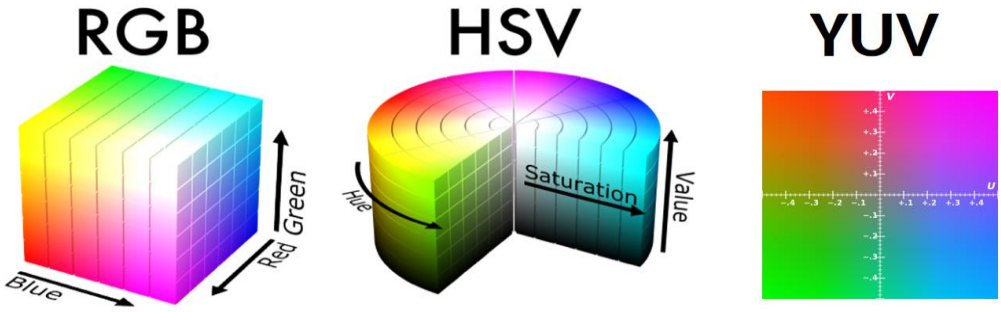


Figure 16: Different color spaces

The RGB representation is the most common color model used in digital imaging. It is often convenient for devices such as displays. The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems. But it is perceptually accurate and there is no easy way to define the properties of a color. The values in RGB range between 0 to 255.

The HSV representation is a different model based on the perception of light. It is a non linear color space composed of three components, Hue defining the color, Saturation defining the amount of color information, and Value which is how much light there is. It allows for easy image transformations, such as shifts in the Hue, and increase in transformation. HSV conversion from RGB images can be achieved by:

$$V_{max} \leftarrow \max(R,G,B)$$

$$V_{min} \leftarrow \min(R,G,B)$$

$$L \leftarrow \frac{V_{max}+V_{min}}{2}$$

$$S \leftarrow \frac{V_{max}-V_{min}}{V_{max}+V_{min}} \text{ if } L < 0.5 \text{ else } \frac{V_{max}-V_{min}}{2-(V_{max}+V_{min})}$$

$$H \leftarrow [60(G-B)/(V_{max}-V_{min})] \text{ if } (V_{max} = R),\ 120+60(B-R)/(V_{max}-V_{min}) \text{ if } (V_{max} = G),$$
$$240+60(R-G)/(V_{max}-V_{min}) \text{ if } (V_{max} = B)$$

If H<0 then H←H+360 . On output 0≤L≤1, 0≤S≤1, 0≤H≤360 .

The YUV model defines a color space in terms of one luma component (Y) and two chrominance components, called U (blue projection) and V (red projection). One important aspect of YUV is that you can throw out the U and V components and get a grey-scale image. Since the human eye is more responsive to brightness than it is to color, many lossy image compression formats throw away half or more of the samples in the chroma channels to reduce the amount of data to deal with, without severely destroying the image quality. Conversion between YUV to RGB and vice versa can be performed as below:

```
R = Y + 1.4075 * (V - 128)
G = Y - 0.3455 * (U - 128) - (0.7169 * (V - 128))
B = Y + 1.7790 * (U - 128)
```

```
Y = R *  .299000 + G *  .587000 + B *  .114000
U = R * -.168736 + G * -.331264 + B *  .500000 + 128
V = R *  .500000 + G * -.418688 + B * -.081312 + 128
```

4.1.5 Image Normalization

Image normalization is a process, often used in the preparation of data sets for artificial intelligence (AI). It is a typical process in image processing that changes the range of pixel intensity values. Its normal purpose is to convert an input image into a range of pixel values that are more familiar or normal to the senses, hence the term normalization. Overall the objective is to have the multiple images put into a common statistical distribution in terms of size and pixel values. The three main types of pixel normalization techniques supported many machine learning frameworks are as follows:

- Pixel Normalization: scale pixel values to the range 0-1.
- Pixel Centering: scale pixel values to have a zero mean.
- Pixel Standardization: scale pixel values to have a zero mean and unit variance.

The pixel standardization is supported at two levels: either per-image (called sample-wise) or per-dataset (called feature-wise). Specifically, the mean and/or mean and standard deviation statistics required to standardize pixel values can be calculated from the pixel values in each image only (sample-wise) or across the entire training dataset (feature-wise).

The most common normalization technique applied in deep/machine learning techniques is the latter, to ensure the mean and the standard deviation to be 0 and 1, respectively. This technique is to re-scale features value with the distribution value between 0 and 1 is useful for the optimization algorithms, such as gradient descent, that are used within machine learning algorithms that weight inputs (e.g., regression and neural networks). Considering an entire dataset of images X, where each image $x_i \; s.t. x_i \in X$ belongs to the standardized version of an image $x_i^{\;s}$ can be retrieved using Eq. 2.

$$x_i^{\;s} = \frac{x_i - mean(X)}{standardeviation\,(X)} \qquad \textbf{(Eq. 2)}$$

The normalization is implemented as show in Eq. 2, in the following python code [3].

```
class MyDataset(Dataset):
    def __init__(self, X, y, transform=None):
        self.data = X
        self.target = y
        self.transform = transform
```

```
def __getitem__(self, index):
    x = self.data[index]
    y = self.target[index]

    # Normalize your data here
    if self.transform:
        x = self.transform(x)

    return x, y

def __len__(self):
    return len(self.data)
```

In this use case, we could set transform to something like this:

```
transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
```

where the **mean** and **std** are scalar values per image channel computed from the dataset.

4.1.6 Duplicate Data Removal

Having duplicate images in your dataset creates a problem for two reasons:

- It introduces bias into your dataset, giving a deep neural network additional opportunities to learn patterns specific to the duplicates
- It hurts the ability of a model to generalize to new images outside of what it was trained on

While we often assume that data points in a dataset are independent and identically distributed, that's rarely (if ever) the case when working with a real-world dataset. When training a Convolutional Neural Network, we typically want to remove those duplicate images before training the model. Trying to manually detect duplicate images in a dataset is extremely time-consuming and error-prone — it also doesn't scale to large image datasets. We therefore need a method to automatically detect and remove duplicate images from our deep learning dataset.

We will apply our hashing function to every image in our dataset. The **dhash()** function handles this calculation to create a numerical representation of the image. When two images have the same hash, they are considered duplicates. With additional logic, we'll be able to delete duplicates and achieve the objective of this project.

The function accepts an image and hashSize and proceeds to:

- Convert the image to a single-channel grayscale image
- Resize the image according to the hashSize. The algorithm requires that the width of the image have exactly 1 more column than the height as is evident by the dimension tuple.
- Compute the relative horizontal gradient between adjacent column pixels. This is now known as the "difference image."

4.1.7 Histogram Equalisation

Histogram equalisation is one of the fundamental image processing operations. The goal of histogram is to improve the overall image quality by suppressing the undesired distortions or enhancing the image features. The histogram equalisation is primarily used for image contrast adjustment and falls into the pixel brightness transformation techniques. The approach can be divided into two branches: a) Global histogram equalisation [4], b) Local histogram equalisation [5].

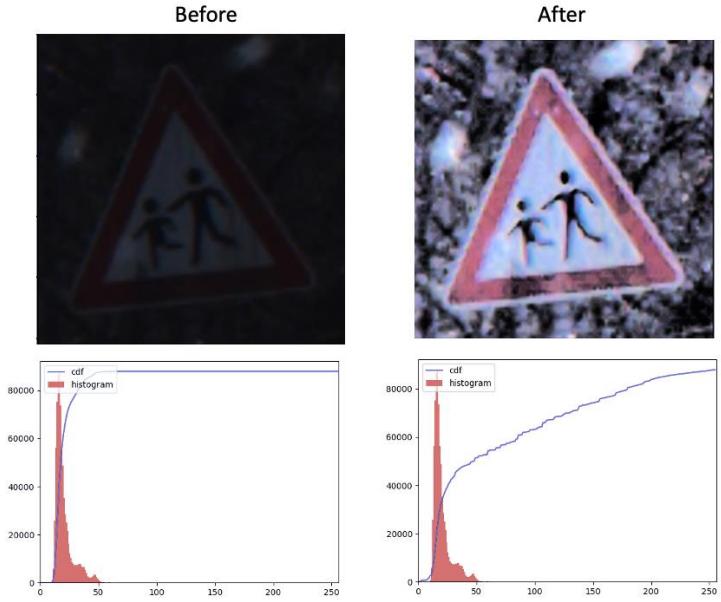


Figure 17: Visualisation of global histogram equalisation. Top row - Captured traffic sign image, bottom row - Histogram and Cumulative Distribution Function (CDF) for top row images. The Left column shows the image before and the right after global histogram equalisation.

a) Global Histogram Equalisation (GHE)
The global histogram equalisation is a simple and fast method however, it is not flexible as local histogram equalisation. In this approach, an image is used as input to compute the histogram transformation function. Hence, the dynamic range of the image histogram is flattened and stretched [4]. Figure 17 shows the effect of global histogram equalisation. In the figure, the top-left traffic sign is hardly visible whereas after the global histogram equalisation, the image at top-right is clear with higher quality. The histogram equalisation function requires a grayscale image and cannot be applied directly to colour images, hence the RGB (Red, Green, Blue) image channels are first converted into HSL (Hue, Saturation, Lightness) format and the equalisation is applied on the lightness channel and then its is converted back to the RGB format.

To calculate the global histogram equalisation, first the histogram of the input image f is calculated:

$$h[i] = \sum_{x=1}^{N} \sum_{y=1}^{M} \begin{cases} 1, & if\ f[x,y] = i \\ 0, & otherwise \end{cases}$$

where, $h[i]$indicates the L uniformly-spaced histogram buckets, L = 2^8 and $M \times N$ are the image dimensions, $f[x,y]$ is a i^{th} pixel value. Next the Cumulative Distribution Function) (CDF) is calculated based on the following equation:

$$CDF[j] = \sum_{i=1}^{j} h[i]$$

CDF is the cumulative distribution function and $h[i]$ is the image histogram. Lastly, the input image is scaled using the CDF to produce the final output.

$$g[x,y] = \frac{CDF\,[F[x,y]] - CDF_{min}}{(N \times M) - CDF_{min}} \times (L-1)$$

where, $g[x,y]$ is the final output image, CDF_{min} is the smallest non-zero value of the cumulative distribution function.

b) Local Histogram Equalisation (LHE)

In local histogram equalisation method, the image is divided into small blocks, and each of these blocks is then histogram equalised. One of the popular local histogram equalisation methods is CLAHE (Contrast Limited Adaptive Histogram Equalisation) [6] and it has been proved that is more effective than the global histogram equalisation approach. The global histogram equalisation only considers the global image contrast and in many cases, it might produce undesirable results. Figure 18, shows the outcome of CLAHE method. In comparison to the global histogram equalisation as shown in Figure 17, the local one produced better results of higher quality. In Figure 18, the image was divided into 8x8 pixel blocks and histogram equalisation was performed on each one of them.

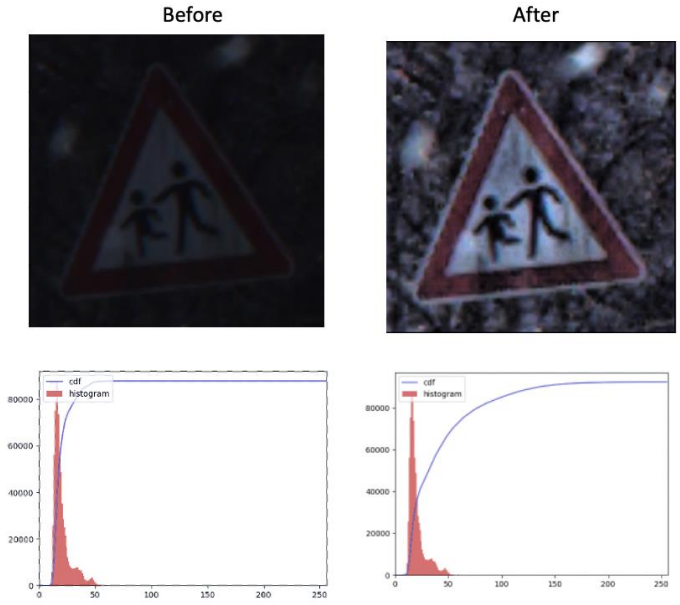


Figure 18: Visualization of CLAHE method [6] i.e., local histogram equalisation. Top row - Captured traffic sign image, bottom row - Histogram and Cumulative Distribution Function (CDF) for top row image. Left column shows the image before equalisation and right after local histogram equalisation.

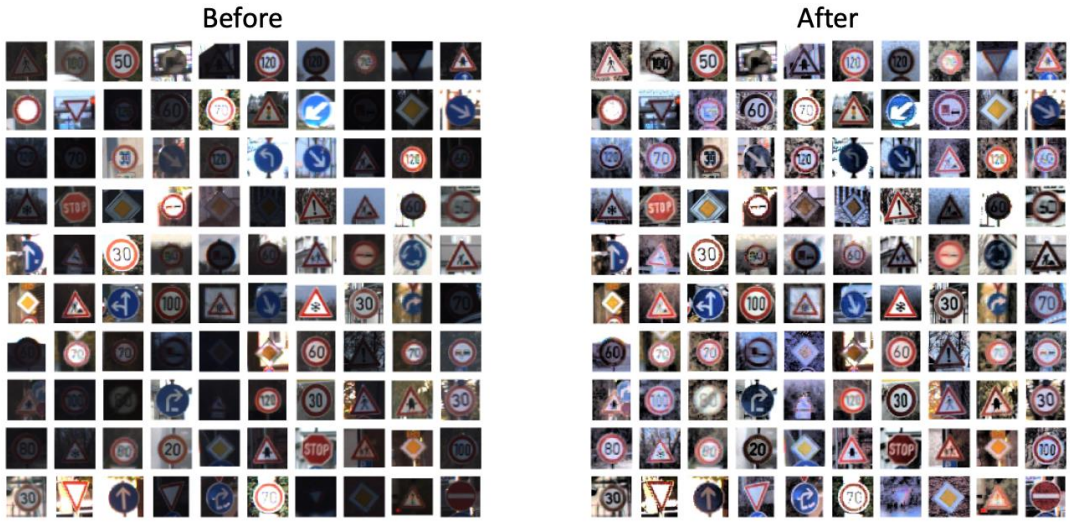


Figure 19: The figure shows the effect of histogram equalisation on the traffic signs. The left images are raw samples from the GTSRB [7] dataset and the right ones are the outcome of histogram equalisation.

4.1.8 Image Cropping

Image cropping is an important part of scene analysis for autonomous vehicles. When images are captured, it might contain information that is not essential for certain tasks, such as traffic sign recognition. Also, in order to reduce the computational complexity of certain processes only a small part of the image data are considered. Hence, image cropping methods can be used to remove unwanted scene content and improve the overall performance [8]. Figure 20 shows the outcome of an image cropping method.

Figure 20: An example of image cropping, here a traffic sign is cropped from the image frame.

In order, to crop a selected region from an image, either two points which define the top-left and bottom-right pixel coordinates of the cropped area or top-left pixel coordinates, width and height of the cropped area are required. In both cases, a rectangle area should be defined.

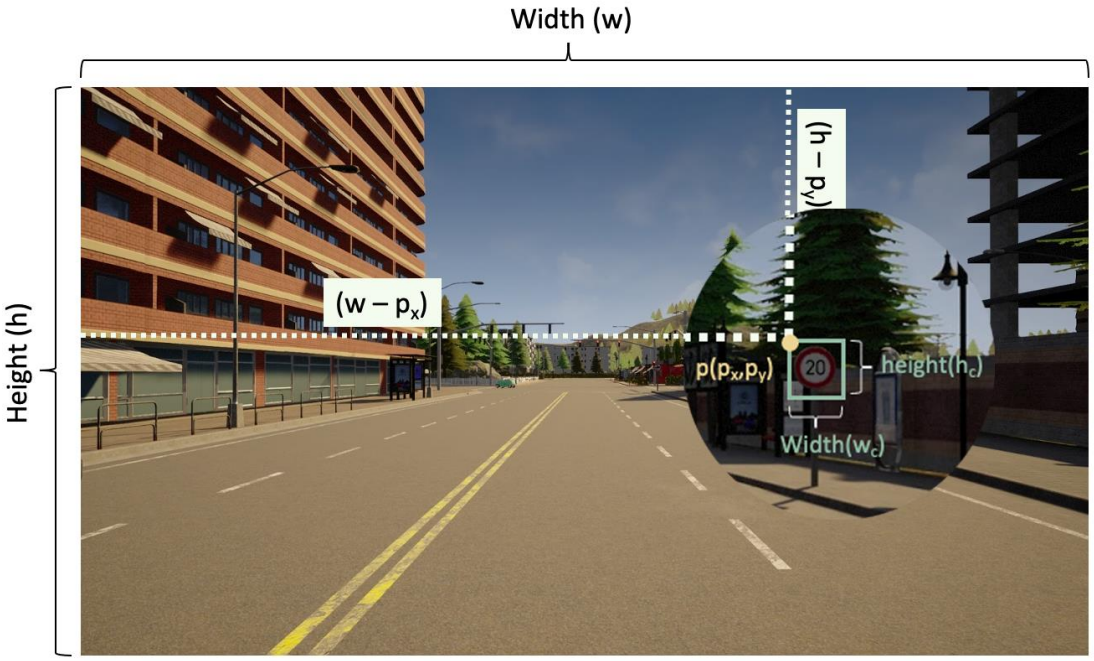


Figure 21: Visualisation of the notation used for the image cropping.

Let i be the image frame with w and h the width and height respectively, then the cropped image i_c with size $w_c \times h_c$ and the top-left pixel coordinates to be (p_x, p_y), then i_c can be calculated from following equation with a visualisation of the all the notion as to be shown in Figure 21:

$$i_c = \int_{x=0}^{x=w} \int_{y=0}^{y=h} i\{x = p_x : p_x + w_c, \quad y = p_y : p_y + h_c\}$$

4.2 Data used for GPS Spoofing Attack

4.2.1 Data Description

The data used for the GPS location spoofing attack detection solution are provided by the CARLA simulator [9]. Note that in the final implementation in the CARAMEL anti-hacking device these data will be provided by the sensors in the autonomous vehicle and will be combined with the signal data pertaining to the Signals-of-Opportunity (SoO) localization solution. The CARLA can provide sensor readings coming from on-board vehicle sensors such as accelerometer [*m/s2*], gyroscope [*rad/s*], and compass [*deg*] included in the Inertial Measurement Unit (IMU). The data from CARLA simulator are provided in a ".CSV" file as shown in Figure 22 where all sensor measurements are sorted over time based on the Timestamp value.

	A	B	C	D	E	F	G	H	I	J
1	Timestamp	Speed	Compass	Gyroscope_x	Gyroscope_y	Gyroscope_z	Geolocation_x	Geolocation_y	Geolocation_z	Steer
2	11.3110	0.0009	180.0000	0.0000	0.0000	0.0000	-1.7000	79.3000	0.0000	0.1000
3	11.3440	0.0007	180.0200	0.0000	0.0000	0.0000	-2.1000	79.7000	0.0000	0.1000
4	11.3780	0.0169	180.0200	0.0000	0.0000	0.0000	-2.1000	80.0000	0.0000	0.1000
5	11.4110	0.0186	180.0200	0.0000	0.0000	0.0000	-2.1000	80.4000	0.0000	0.0000
6	11.4440	0.0204	180.0200	0.0000	0.0000	0.0000	-2.1000	80.7000	0.0000	0.0000
7	11.4780	0.0220	180.0200	0.0000	0.0000	0.0000	-2.1000	81.1000	0.0000	0.0000
8	11.5110	0.0236	180.0200	0.0000	0.0000	0.0000	-2.1000	81.4000	0.0000	0.0000
9	11.5440	0.0252	180.0200	0.0000	0.0000	0.0000	-2.1000	81.8000	0.0000	0.0000
10	11.5780	0.0267	180.0200	0.0000	0.0000	0.0000	-2.1000	82.1000	0.0000	0.0000
11	11.6110	0.0281	180.0200	0.0000	0.0000	0.0000	-2.1000	82.5000	0.0000	0.0000
12	11.6440	0.0295	180.0200	0.0000	0.0000	0.0000	-2.1000	82.8000	0.0000	0.0000
13	11.6780	0.0308	180.0200	0.0000	0.0000	0.0000	-2.1000	83.2000	0.0000	0.0000
14	11.7110	0.0322	180.0200	0.0000	0.0000	0.0000	-2.1000	83.5000	0.0000	0.0000
15	11.7440	0.0334	180.0200	0.0000	0.0000	0.0000	-2.1000	83.9000	0.0000	0.0000
16	11.7780	0.0346	180.0200	0.0000	0.0000	0.0000	-2.1000	84.2000	0.0000	0.0000
17	11.8110	0.0358	180.0200	0.0000	0.0000	0.0000	-2.1000	84.6000	0.0000	0.0000
18	11.8440	0.0369	180.0200	0.0000	0.0000	0.0000	-2.1000	84.9000	0.0000	0.0000
19	11.8780	0.0380	180.0200	0.0000	0.0000	0.0000	-2.1000	85.3000	0.0000	0.0000
20	11.9110	0.0391	180.0200	0.0000	0.0000	0.0000	-2.1000	85.6000	0.0000	0.0000
21	11.9440	0.0401	180.0200	0.0000	0.0000	0.0000	-2.1000	86.0000	0.0000	0.0000

Figure 22: Data structure provided from CARLA simulator.

Referring to the detection method diagram shown inFigure 23, the vehicle's state is predicted using the vehicle state model (i.e., bicycle model, as described in D4.3) and the following in-vehicle sensor data:

- Compass value [*deg*]; is used to estimate the heading rate of the vehicle.
- Gyroscope value [*rad/s*]; is used to calculate the yaw rate.
- Speed value [*m/s*]; is used to estimate the velocity component of the vehicle.
- Steering value [*deg*]; is used to measure the steering angle (left or right) of the vehicle.
- Timestamp value [*ms*]; is used for processing synchronization, in terms of the length of the data to be processed.

In the comparison step, CARLA provides GNSS data from the GNSS sensor module which is used to compare the fall-back location solution with the actual GPS readings of the vehicle. Specifically, CARLA provides Geolocation data [*m*] (x,y,z) coordinates in order to obtain the current position of the vehicle.

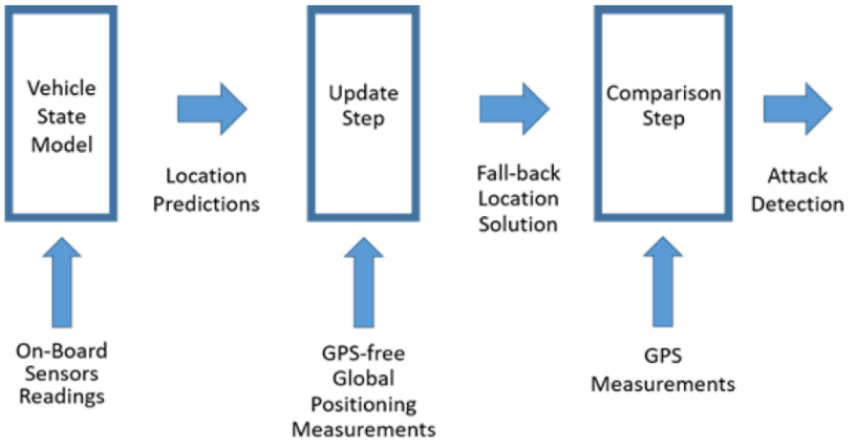


Figure 23: GPS Location Spoofing Detection algorithm.

4.2.2 Data Processing

As mentioned in the **D4.3**, the comparison step for the in-vehicle GPS location spoofing attack detection can be performed using the Bhattacharyya distance. Under the assumption that the GPS receiver can provide together with the GPS location an estimated error (in meters) which can be transformed into a covariance matrix, the Bhattacharyya distance can compute the amount of overlap of two statistical distributions. In order to reduce the number of false alarms due to the spurious GPS measurement errors, the Bhattacharyya distance is averaged over a sliding window of W seconds:

$$D = \sum_{n\in[k-W,k]} \frac{1}{8}\mu[n]\left(\frac{\Sigma_{x,y}[n]+\Sigma^{G}_{x,y}[n]}{2}\right)^{-1}\mu[n]^{T} + \frac{1}{2}\ln\left(\frac{\det\frac{\Sigma_{x,y}[n]+\Sigma^{G}_{x,y}[n]}{2}}{\sqrt{\det\Sigma_{x,y}[n]\,\det\Sigma^{G}_{x,y}[n]}}\right)$$

where $\mu[n] = \left(\mu_x[n],\ \mu_y[n]\right) - \left(\mu^{G}_x[n], \mu^{G}_y[n]\right)$ and $[\mu_x, \mu_y]$, $\Sigma_{x,y}$ and $[\mu^{G}_x, \mu^{G}_y]$, $\Sigma^{G}_{x,y}$, are the vehicle and GPS location estimates and their covariance matrices, respectively.

The impact of the sliding window on the detection performance is shown in Figure Figure 24. The analysis shows the detection rate of the algorithm in terms of the attack bias introduced in the GPS measurement and the length of the applied sliding window. Clearly, the increase of the window length improves significantly the detection performance, especially for relatively small attack bias.

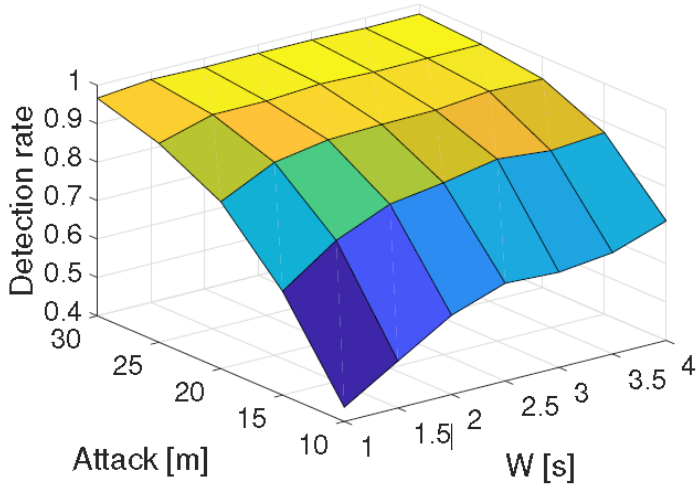


Figure 24: ***Attack detection rate in terms of window length and attack bias***

4.3 Data used for V2X messages Tracking Attack

4.3.1 Data Description

The Cologne dataset and the dataset of the Caramel V2X messages are compared in Table 14. The first difference is that the simulated dataset of the Cologne city is updated every second for all the vehicles, and the V2X messages dataset is updated randomly (periods of 100 ± 50 ms) and differently for each vehicle. This needs to be in this way because the vehicles are not synchronized when sending the V2X messages and the sent period of each vehicle is not guaranteed to be constant.

Table 14: Cologne Dataset and Caramel V2X messages Dataset comparison

Cologne Dataset (Uppor, 2014)	Caramel V2X messages Dataset
Data update period ● The same for all the vehicles ● 1 second Variables ● Vehicle position x and y ● Vehicle velocity module v	Data update period: ● Different for each vehicle ● 100 ± 50 ms Variables ● Vehicle position x and y ● Vehicle velocity module v, and velocity components v_x and v_y ● Acceleration

Table 15: Dataset content sample

n	id	time [s]	x [m]	y [m]	v [m/s]
0	1	0	18.385	19.396	13,83
1	1	1	18.396	19.389	13,82
2	1	2	18.408	19.381	13,47
3	2	0	18.259	19.142	21,56
4	2	1	18.270	19.124	21,42
5	2	2	18.282	19.105	21,64
6	3	0	18.314	19.054	0,17
7	3	1	18.314	19.054	0,02
8	3	2	18.314	19.054	0,01

Table 16: Dataset content sample

n	id	time [s]	x [m]	y [m]	v [m/s]	v_x [m/s]	v_y [m/s]
0	1	0	18.385	19.396	13,83	11,62	-7,50
1	1	0,13	18.386	19.395	13,83	11,63	-7,48
2	1	0,21	18.387	19.395	13,82	11,64	-7,46
3	1	0,29	18.388	19.394	13,82	11,64	-7,44
4	1	0,43	18.389	19.393	13,82	11,65	-7,43
5	2	0,02	18.259	19.142	21,56	11,37	-18,31
6	2	0,11	18.260	19.140	21,50	11,26	-18,32
7	2	0,18	18.262	19.138	21,46	11,17	-18,33
8	2	0,32	18.263	19.137	21,43	11,09	-18,34
9	2	0,41	18.264	19.135	21,40	11,03	-18,34
10	3	0,02	18.314	19.054	0,17	0,10	-0,14
11	3	0,12	18.314	19.054	0,11	0,07	-0,09
12	3	0,17	18.314	19.054	0,07	0,04	-0,05
13	3	0,33	18.314	19.054	0,03	0,02	-0,02
14	3	0,44	18.314	19.054	0,00	0,00	0,00

4.3.2 Data Processing

Two operations need to be performed to create the V2X dataset from the Cologne dataset:

i) First of all, it is needed to calculate the v_x and v_y components of the velocity v for all the n positions.

ii) And second, it is necessary to do the interpolation of the position x and y and the velocity components v_x and v_y for any time in which the vehicle is in between the positions n and n+1.

The v_x and v_y components of the velocity v have been calculated by multiplying the module of the velocity v (given in the Cologne dataset) by the sinus and cosinus of the angle of the secant line to the curve (see Figure 3). The resultant mathematical expression to calculate $v_{x,n}$ and $v_{y,n}$ are given in equations *Eq. 3* and *Eq 4*.

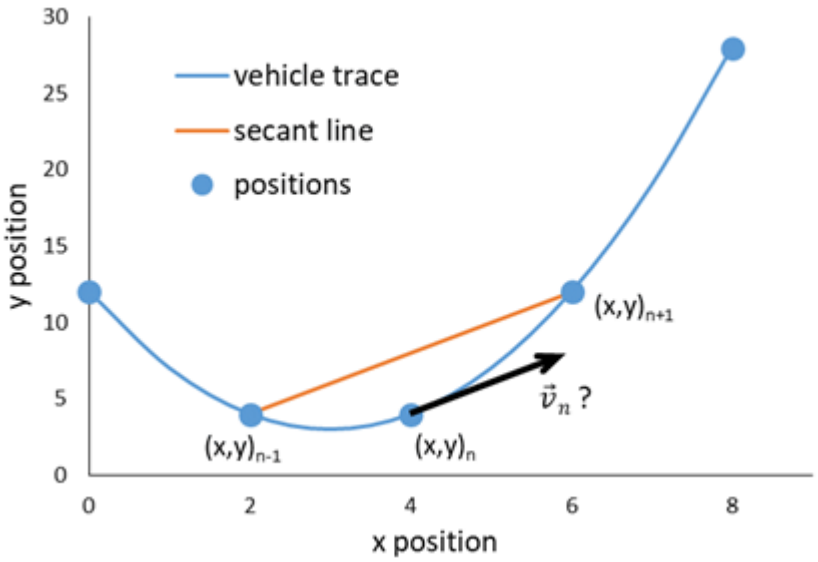


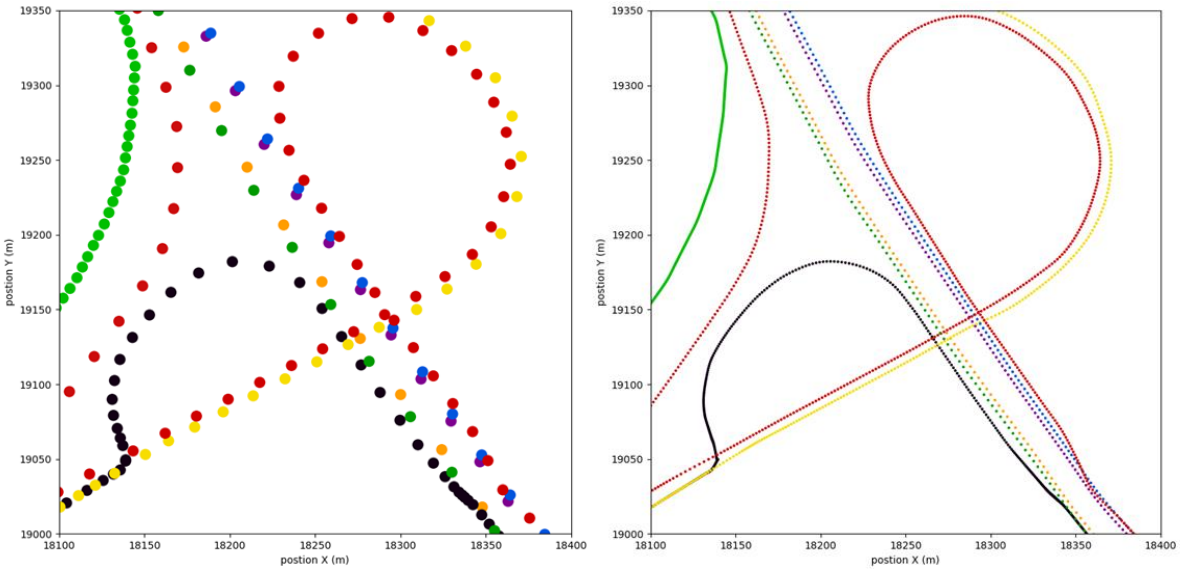
Figure 3

$$v_{x,n} = v_n \cdot \frac{x_{n+1}-x_{n-1}}{\sqrt{(x_{n+1}-x_{n-1})^2+(y_{n+1}-y_{n-1})^2}} \quad \text{[Eq. 3]}$$

$$v_{y,n} = v_n \cdot \frac{y_{n+1}-y_{n-1}}{\sqrt{(x_{n+1}-x_{n-1})^2+(y_{n+1}-y_{n-1})^2}} \quad \text{[Eq. 4]}$$

The interpolation of the position and the velocity in between positions n and n+1 has been done by assuming that the vehicle follows a trajectory of the form of *Eq. 4* (in the direction *x*) and *Eq. 10* (in the direction *y*) in between the points *n* and *n+1*, in which a is the acceleration and j is the jerk.

$x_t = x_n + v_{x,n} \cdot t + \frac{1}{2} a_{x,n} \cdot t^2 + \frac{1}{6} j_{x,n} \cdot t^3$ [Eq. 4]	$y_t = y_n + v_{y,n} \cdot t + \frac{1}{2} a_{y,n} \cdot t^2 + \frac{1}{6} j_{y,n} \cdot t^3$ [Eq. 10]
$v_{x,t} = v_{x,n} + a_{x,n} \cdot t + \frac{1}{2} j_{x,n} \cdot t^2$ [Eq. 5]	$v_{y,t} = v_{y,n} + a_{y,n} \cdot t + \frac{1}{2} j_{y,n} \cdot t^2$ [Eq. 11]
$x_t(t=0) = x_n$ [Eq. 6]	$y(t=0) = y$ [Eq. 12]
$x_t(t=1) = x_{n+1}$ [Eq. 7]	$y_t(t=1) = y_{n+1}$ [Eq. 13]
$v_{x,t}(t=0) = v_{x,n}$ [Eq. 8]	$v_{y,t}(t=0) = v_{y,n}$ [Eq. 15]
$v_{x,t}(t=1) = v_{x,n+1}$ [Eq. 9]	$v_{y,t}(t=1) = v_{y,n+1}$ [Eq. 16]



4.4 Data used for Attack on Electric charging stations

ChargePoints communicate with their central system according to messages from the OCPP protocol. Each chargepoint holds a sim card. The sim connects to the provider through an APN, a private network separate from the rest of the internet. The provider aggregates all channels from chargepoints. From the provider there is a single encrypted connection (VPN tunnel) with GreenFlux's Azure Virtual Network which at the end contains the WebSocketServer. The WebSocketServer makes sure the connections with the chargepoints remain intact. The WSS talks to the service bus over a HTTPS connection. The figure below shows a schematic overview of GreenFlux's communication architecture.

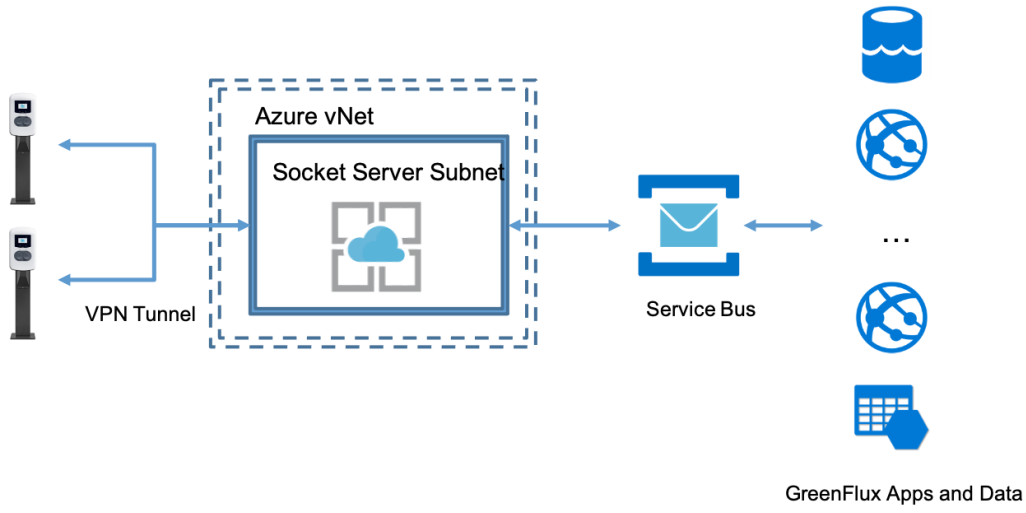


Figure XX: GreenFlux's Communication Architecture (simplified)

The servicebus talks to individual services that either read or write data. There are separate services to handle for instance transaction requests or store measurements coming from chargepoints.

Messages handled by the WebSocketServer are stored for a period of 30 days. Data stored by the services is usually stored permanently. The next chapter describes data from the WebSocketServer and GreenFlux databases in more detail.

4.4.1 Data Description

All incoming and outgoing messages go through the web socket server. The messages that go through the WSS are stored for a period of 30 days. Every request (CALL) that is sent by either the server or the client must have a reply (CALLRESULT). Or an explanation (CALLERROR) in case the message cannot be processed.

Message Type

In order to identify messages, they must always be provided with a message type number as shown in the table below:

Table 17: Message types

MessageType	MessageTypeNumber	Remarks
CALL	2	Request
CALLRESULT	3	Response to specific request
CALLERROR	4	Error response to specific request

Messages with a MessageTypeNumber other than listed in the table above are ignored by the server.

Message ID

A message ID is used to identify unique requests. For a CALL message, the message ID must be different from all previously used message IDs by the same sender with the same MessageType. For a CALLRESULT or CALLERROR, the messageID must be the same as that of the corresponding request.

Table 18: Message IDs

Name	Datatype	Restrictions
messageId	string	Maximum 36 characters

Call

A message sequence always starts with a call. This consists of 4 elements, namely: MessageTypeId, UniqueId, required Action and a Payload (associated with the action). The syntax is as follows:

[<MessageTypeId>, "<UniqueId>", "<Action>", {<Payload>}]

The meaning of these elements is shown in the table below:

Table 19: Call elements

Element name	Meaning
UniqueId	Identifier used to match request with response
Action	Name of the remote procedure or action
Payload	JSON object containing the actual content of the message

An example message with a StartTransaction.request may then look as follows;

```
[2,"5aec86526bca5L","StartTransaction",{"connectorId":1,"meterStart":687476
0,"idTag":"62a004b95946","timestamp":"2020-06-08T07:36:36Z"}]
```

CallResults

When the call has been processed, a call result follows. These always consist of 3 elements: MessageTypeId, UniqueId and a payload (which contains the response to the Action in the original call). The syntax of a CallResult is as follows:

[<MessageTypeId>, "<UniqueId>", {<Payload>}]

The meaning of each element is explained in the table below:

Table 20: CallResult elements

Element name	Meaning
UniqueId	Identifier to match the CallResult with the original Call. The ID must be exactly the same.
Payload	JSON object containing the actual content of the message. In this case the result of the action that was performed.

An example StartTransaction.response may look like:

```
[3,"5aec86526bca5L",{"idTagInfo":{"status":"Accepted","expiryDate":"2020-
06-16T09:14:59Z","parentIdTag":""},"transactionId":10198219}]
```

Frequently used messages

OCPP message types that are processed by the WSS are shown in the table below:

Table 21: Frequently used OCPP Message types

MessageType	Initiated by	Meaning	Dedicated Service
Authorization	ChargePoint	ChargePoint has to authorize owner of EV before starting or stopping transaction	Transactions Service
StartTransaction	ChargePoint	Inform central system that a transaction has started	Transactions Service
StopTransaction	ChargePoint	Inform central system that a transaction has stopped	Transactions Service
SetChargingProfile	Central system	SetChargingProfiles are used by the central system to send *ChargingProfiles* to chargers. These messages are used in the context of Smart Charging	Command Service
MeterValues	ChargePoint	MeterValues contain in session meter readings of the ChargePoint's internal (electrical) meter.	Transactions Service
HeartBeat	ChargePoint	HeartBeats are sent by ChargePoint to let the central system know it is still online	Status Service
StatusNotification	ChargePoint	ChargePoint notifies central system about status change or error within ChargePoint	Status Service

4.4.2 Data Processing

The service bus translates OCPP messages into so-called GCPP (GreenFlux internal language for processing OCPP messages). From here, messages are routed on to their dedicated service. Parts of these services are described in more detail in the sections below.

Transaction Service

This services handles all transaction related messages. It has the following functionality:

- Process Authorize messages from Charge Stations, which are received via the GSOP Socket Server. It will authenticate the token against the Token Service, and will respond to Charge Point and send an updated message to GSOP.

  Authorize requests contain:
  - idTag
- Process StartTransaction messages from Charge Stations, which are received via the GSOP Socket Server. It will authenticate the transaction against the Token Service, and will create a transaction document.

StartTransaction requests contain:
- connectorId
- idTag
- meterStart
- timeStamp

- Process StopTransaction messages from Charge Stations, which are received via the GSOP Socket Server. It will authenticate the transaction against the Token Service, and will complete a transaction, and send a message to GSOP.

  StopTransaction requests contain:
  - meterStop
  - timestamp
  - transactionId
  - reason
- Process Meter Value messages from Charge Stations, which are received via the GSOP Socket Server. It will process the meter values, update the charging period, and send a message to GSOP.

  MeterValue requests contain:
  - connectorId
  - transactionId
  - sampledValue
    - value
    - context
    - format
    - measurand
    - phase
    - location
    - unit

Status Service
Purpose of this service:

- Save and Update Connectivity Status based on missed HeartBeat messages. If no activity happened on the ChargePoint and no HeartBeat messages are received by the central system for an interval longer than 90 min it is assumed that the charging station is offline. There is no content defined for HeartBeat requests.
- Save and Update Status of Charge Point based on StatusNotification messages.

  StatusNotification requests contain:
  - connectorId
  - errorCode
  - info
  - status
  - timestamp
  - vendorId
  - vendorErrorCode

Command Service

Commands that are sent to ChargePoints are processed by the Command Service. Within the context

of CARAMEL, especially messages used for smart charging are relevant: the so-called *SetChargingProfile* messages. The smart charging mechanism determines at connector level what the capacity is for a specific time interval. The intelligence of this is a separate part of the GreenFlux platform, but it uses the CommandService to send the correct messages to the Charge Point.

SetChargingProfile requests contain:

- connectorId
- ChargingProfile
    - ChargingProfileId
    - transactionId
    - stackLevel
    - chargingProfilePurpose
    - chargingProfileKind
    - recurrencyKind
    - validFrom
    - validTo
    - chargingSchedule
        - duration
        - startSchedule
        - schedulingUnit
        - chargingSchedulePeriod
            - startPeriod
            - limit
            - numberPhases

5 Conclusion

This document has outlined details on various aspects of the data collection and pre-processing steps employed in CARAMEL and are intended for use within the antihacking device. These details included the architecture, access protocols and front-end information for the data generated by the autonomous demo vehicle. We also outline complementary data generation approaches such as the CARLA simulator that allow for quick experimentation. In addition, we describe data formats and data generation related to V2X communications and electric charging stations, as well as various techniques used for preprocessing camera data and GPS data for the autonomous vehicle use case. This preparatory work will enable the easier integration of these techniques in the antihacking device component.

References

[1] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[2] Rafael C. Gonzalez and Richard E. Woods. 2001. Digital Image Processing (2nd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[3] Paszke, Adam and Gross, Sam and Massa, Francisco and Lerer, Adam and Bradbury, James and Chanan, Gregory and Killeen, Trevor and Lin, Zeming and Gimelshein, Natalia and Antiga, Luca and Desmaison, Alban and Kopf, Andreas and Yang, Edward and DeVito, Zachary and Raison, Martin and Tejani, Alykhan and Chilamkurthy, Sasank and Steiner, Benoit and Fang, Lu and Bai, Junjie and Chintala, Soumith, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", Advances in Neural Information Processing Systems, 2019.

[4] H. Lidong, Z. Wei, W. Jun, and S. Zebin, 'Combination of contrast limited adaptive histogram equalisation and discrete wavelet transform for image enhancement', IET Image Processing, vol. 9, no. 10, pp. 908–915, 2015, doi: https://doi.org/10.1049/iet-ipr.2015.0150.

[5] N. M. Kwok, Q. P. Ha, G. Fang, A. B. Rad, and D. Wang, 'Color image contrast enhancement using a local equalization and weighted sum approach', in 2010 IEEE International Conference on Automation Science and Engineering, Aug. 2010, pp. 568–573, doi: 10.1109/COASE.2010.5584719.

[6] K. Zuiderveld, 'Contrast limited adaptive histogram equalization', Graphics gems, pp. 474–485, 1994.

[7] Stallkamp, Johannes, Marc Schlipsing, Jan Salmen, and Christian Igel. 'The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition'. In The 2011 International Joint Conference on Neural Networks, 1453–60. IEEE, 2011.

[8] J. Yan, S. Lin, S. B. Kang, and X. Tang, 'Learning the Change for Automatic Image Cropping', in 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, Jun. 2013, pp. 971–978, doi: 10.1109/CVPR.2013.130.

[9] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: Conference on Robot Learning, pp. 1{16 (2017)