D3.1 Algorithms for monitoring the user and analyzing the scene by fusioning multimodal data

Authors ISI, I2CAT, ATOS, PASEU, 8BELLS, USI, TAU, UoP, CATALINK, RTC, UPAT

Work Package WP3 - Model based CP(H)S Layer Design and Development supporting Distributed Assisted, Augmented and Autonomous Intelligence

Abstract

The overall aim of this deliverable is to develop effective methods for monitoring important aspects of the human activity, behavior and the CP(H)S environment, through the diverse sensing modalities that can be utilized based on the open middleware. To this end, advanced machine learning methods will be investigated so as to translate multimodal data derived from the middleware into meaningful representations of user behavior, and environment status. While the first step will be to understand rather abstract user activities, the second will be to connect these to the user's context and understand in more detail what the user is doing at that time. This will be addressed by employing a hierarchical approach, where user activity and behavior and different expected or unforeseen events, will be treated at gradually increasing levels of granularity, connecting in essence the user low-level actions to their context and employing increasing sensing modalities, towards meaningful representations of user behavior and the environmental conditions.

Funded by the Horizon 2020 Framework Program of the European Union

Deliverable Information

Work Package	WP3- Model based CP(H)S Layer Design and Development supporting Distributed Assisted, Augmented and Autonomous Intelligence
Task	T3.1. CP(H)S medium: Enabling Multimodal Sensing and Embedded Assisted and Augment Intelligence
Deliverable title	Algorithms for monitoring the user and analyzing the scene by fusioning multimodal data
Dissemination Level	PU
Status	Final
Version Number	1.0
Due date	M24

Project Information

Project start and duration	01/01/2020 – 31/12/2022, 36 months
Project Coordinator	Industrial Systems Institute, ATHENA Research and Innovation Center 26504, Rio-Patras, Greece
Partners	1. ATHINA-EREVNITIKO KENTRO KAINOTOMIAS STIS TECHNOLOGIES TIS PLIROFORIAS, TON EPIKOINONION KAI TIS GNOSIS (ISI) the Coordinator 2. FUNDACIO PRIVADA I2CAT, INTERNET I INNOVACIO DIGITAL A CATALUNYA (I2CAT) 3. IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD (IBM ISRAEL) 4. ATOS SPAIN SA (ATOS) 5. PANASONIC AUTOMOTIVE SYSTEMS EUROPE GMBH (PASEU) 6. EIGHT BELLS LTD (8BELLS) 7. UNIVERSITA DELLA SVIZZERA ITALIANA (USI), 8. TAMPEREEN KORKEAKOULUSAATIO SR (TAU) 9. UNIVERSITY OF PELOPONNESE (UoP) 10. CATALINK LIMITED (CATALINK) 11. ROBOTEC.AI SPOLKA Z OGRANICZONA ODPOWIEDZIALNOSCIA (RTC) 12. CENTRO RICERCHE FIAT SCPA (CRF) 13. PANEPISTIMIO PATRON (UPAT)
Website	www.cpsosaware.eu

Control Sheet

VERSION	DATE	SUMMARY OF CHANGES	AUTHOR
0.1	*01/12*	*Initial draft*	*ISI, UPAT, CATALINK, UoP*
0.2	*10/12*	*Pre-final draft*	*ISI, UPAT, CATALINK, UoP*
0.3	*12/12*	*Finalization of the algorithmic Descriptions*	*ISI, UPAT, CATALINK, UoP*
0.4	*14/12*	*Finalization of the Results*	*ISI, UPAT, CATALINK, UoP*
0.5	*20/12*	*Deliverable Reviewed*	*UPAT, IBM*
1.0	*27/12*	*Final Version Ready for Submission*	*ISI*

	NAME
Prepared by	ISI
Reviewed by	UPAT, IBM
Authorized by	ISI

DATE	RECIPIENT
28/12	Coordinator
31/12	European Commission

Table of Contents

List of Figures

List of Tables

Executive Summary

The goal of this deliverable is threefold: i) describe the progress made for the design and implementation for user-state monitoring though multi-modal processing, ii) analyze a variety of multi-modal scene understanding, localization and mapping techniques that were deployed using CARLA simulator, and iii) present novel methods for accelerating DNNs that are explicitly focus on scene analysis and understanding. More specifically:
In Section 2, three approaches for user state-monitoring are discussed. In particular:
1) A fast and reliable facial Driver State Monitoring (DSM) approach was implemented from UPAT. The approach uses the captured data from a single camera for the estimation of the driver's drowsiness status, based on i) the yawing frequency, ii) the eye aspect ratio and iii) the percentage of the eyes closure (blinking). This DSM module captures frontal facial images of the driver to assess fatigue levels based on the activity of his/her eyes via facial analysis of some identified landmarks. The UPAT' DSM approach is mainly used for a real-time implementation while a user drives in the CARLA simulator under different traffic conditions.
2) Catalink has developed a DSM Android application which provides real-time monitoring and assessment of the driver's drowsiness and attention levels. The application uses the smartphone's front camera to monitor the general status of the driver during the driving session and to estimate his/her fatigue and distraction levels.
3) University of Peloponnese has developed a DSM application implementation on Field Programmable Gate Arrays (FPGA) for monitoring the driver's drowsiness. It is based on the open-source software package Deformable Shape Tracking (DEST). This approach uses 68 landmarks that are aligned on the face detected in a new frame using an Ensemble of Regression Trees (ERT). The original DEST package was only tested on Windows platforms thus, initially it was ported to Ubuntu environment where variations and extensions on the source code can be tested easily: fast project rebuild, advanced debugging facilities are available through GDB and there is feedback from real time output video. The source code structure of the Ubuntu DEST implementation is directly portable to Xilinx Vitis environment for hardware implementation of the most computationally intensive operations.
As an additional contribution regarding the manufacturing pillar, we note that in industrial environments the pose

recognition of operators is important for relevant tasks such as for increasing the operator's situational awareness during human-robot interaction, for the safe training of new operators, for ergonomics analysis of the operator under different actions during his/her collaboration with a robot etc. In the CPSoSaware project, we tested algorithms for the landmarks extraction from human postures using the captured activity of a human from a camera and evaluated the accuracy and performance in different parameterization settings of the implemented pose estimation approach. Furthermore, assuming that there are some static cameras in different locations of the manufacturing environment, the CPSoSaware implementation estimates in real-time the confidence rate of the operator's pose while he/she performs collaborative actions with the robot in the manufacturing environment. Additionally, since some tasks need more emphasis to be given in specific areas of the operator's body, the developed algorithm allow the selection of specific points of interest (e.g., above the torso), related to the operator's task that we want to monitor and analyze.

Section 3 presents four multi-modal DNN based methods for object detection in three scenes: SqueezeDet, ResNet, Pointpillar and PVRCNN. Both SqueezeDet and ResNet exploit 2D images, while Pointpillar and PVRCNN rely on 3D point clouds. Additionally, three multi-modal SLAM techniques (ORB-SLAM, DSO and Lego-LOAM) which have been integrated in the CARLA simulator are analyzed.

Deep learning and, especially, the use of DNNs provides impressive results in analyzing and understanding complex and dynamic scenes from visual data. The prediction horizons for those perception systems are very short and inference must often be performed in real time, stressing the need of transforming the original large pre-trained networks into new smaller models, by utilizing Model Compression and Acceleration (MCA) techniques. Thus, the goal of Section 4 is to investigate best practices for appropriately applying novel weight sharing techniques, optimizing the available variables and the training procedures towards the significant acceleration of widely adopted DNNs. Extensive evaluation studies carried out using various state-of-the-art DNN models in object detection and tracking experiments, provide details about the type of errors that manifest after the application of weight sharing techniques, resulting in significant acceleration gains with negligible accuracy losses.

1 Introduction

The main outcomes of this deliverable related to user-state monitoring, multi-modal processing for scene analysis, localization and mapping, as well as accelerating scene understanding DNNs, have been identified below:
Section 2 presents three different user-state monitoring approaches, developed by UPAT, CATALINK and UoP for DSM. UPAT's solution is mainly used for a real-time implementation while a user drives in the CARLA simulator under different traffic conditions. CATALINK has developed a DSM Android application which provides real-time monitoring and assessment of the driver's state. UoP has developed a DSM application implemented on FPGA. Below, we provide the code and video links for the discussed demos:

1. Code link: https://gitlab.com/isi_athena_rc/cpsosaware/drowsiness-detection
2. Video link: https://drive.google.com/file/d/1I0ieo_vTT2owSTHQ-LZa8J6iQjVD4wUl/view?usp=sharing

Section 3 discusses four multi-modal scene understanding solutions based on DNNs and three multi-modal SLAM methods deployed in CARLA simulator. Below, we provide the code and video links for the discussed demos:

1. Code link: https://gitlab.com/isi_athena_rc/cpsosaware/odometers
2. Video link:
   - https://drive.google.com/file/d/1_KsEccdora1KFkkVfBDg7cO2wLXv5rSA/view?usp=sharing ,
   - https://drive.google.com/file/d/1SpsiwKUexpzIHXYbJPTfiJc2HDirMZJI/view?usp=sharing ,
   - https://drive.google.com/file/d/1nkpJFLEF_VETiFfRpfOwZiaVIQBVxyQd/view?usp=sharing ,
   - https://drive.google.com/file/d/1YKuP_y-sipBwITGZTlHOkdvLhfj_hVhm/view?usp=sharing ,
   - https://drive.google.com/file/d/16qWFddS_91eGGLo5nvQQ_P_2240I4e9l/view?usp=sharing

Section 4 is dedicated to the acceleration of scene analysis and understanding DNNs, describing novel solutions. Below, we provide the code link:

1. Code link: https://gitlab.com/isi_athena_rc/cpsosaware/multimodal-scene-understanding/multimodal-scene-understanding-with-model-acceleration

2. Video Link :
   - https://drive.google.com/file/d/1jRC6EYMxUKW54PmYgBXAkVJWU_7KC1-U/view?usp=sharing
   - https://drive.google.com/file/d/1SpsiwKUexpzIHXYbJPTfiJc2HDirMZJI/view?usp=sharing

2 Multi-modal processing for user state monitoring

2.1 Driver state monitoring

2.1.1 UPAT solution for driver

In the literature, there are mainly three ways of detecting driver's drowsiness, categorized as follows: (i) the physiological changes in the body (e.g., pulse rate, heart activity), which can be detected by wearable sensors or bracelet systems, (ii) the behavioral measures (e.g., eye closure, blinking, yawning, posture analysis), which is achieved by eye-tracking, pose recognition and graph-based drowsiness algorithms, and (iii) the vehicle-based measurements (e.g., lane position, steering wheel movements) that use measurements of the same vehicle. For the purposes of task T3.1, the proposed DSM implementation utilizes a camera to capture the driver's face and then an algorithm is used to analyze the face in order to extract facial landmarks. Next, landmarks of eyes and mouth will be used to identify the driver's drowsiness.

Regarding the process of face analysis (e.g., face identification, landmarks extraction, etc), the dlib library[1] is utilized. The dlib library is without a doubt one of the most utilized packages in many applications, due to the high efficiency and the robustness of its results as well as the fast time performance. Then, a Python package[2] wraps dlib's face recognition functions into a simple, easy to use API. The dlib library includes two methods that can be used for face detection:

1. A Histogram of Oriented Gradients (HOG) + Linear SVM face detector (which is an accurate and computationally efficient approach).
2. A Max-Margin (MMOD) CNN face detector (which is highly accurate and very robust, capable of detecting faces from varying viewing angles, lighting conditions, and occlusion). Additionally, it is worth mentioning that the MMOD face detector can run on an NVIDIA GPU, making it even faster.

Facial landmarks are used to localize and represent salient and meaningful regions of the face, such as the eyes, eyebrows, nose, mouth, jawline. Detecting facial landmarks can be assumed as a subset of the shape prediction problem. Given an input image/frame (and normally a region-of-interest that specifies the object of interest), a shape predictor attempts to localize landmarks of interest along with the shape. In the context of facial landmarks, the main objective is to detect important facial structures on the face using shape prediction methods. Detecting facial landmarks is, therefore, a two-step process:

Step 1: Identify the face in the image/frame.

Step 2: Detect the key facial structures on the face region-of-interest.

In literature, there are several ways that the step 1 can be accrued, such as:

- To use the OpenCV's built-in Haar cascades.
- To apply a pre-trained HOG + Linear SVM object detector specifically for the task of face detection.
- To use deep learning-based algorithms for face localization.

Given the region of the face that is acquired by step 1, step 2 can be then applied. There are also a variety of facial landmark detectors, however, all methods essentially try to localize and label the following facial regions:

- Mouth
- Right eyebrow

[1] http://dlib.net/

[2] https://pypi.org/project/dlib/

- Left eyebrow
- Right eye
- Left eye
- Nose
- Jaw

The facial landmark detector, which is included in the dlib library, is an implementation of the work presented in the paper[3]. The method starts by using: (i) A training set of labelled facial landmarks on an image. These images are manually labelled, specifying specific (x, y)-coordinates of regions surrounding each facial structure. (ii) Priors, or in other words, the probability of distance between pairs of input pixels.
Given the training data, an ensemble of regression trees is trained to estimate the facial landmark positions directly from the pixel intensities themselves. The result is the design of a facial landmark detector that is capable to detect facial landmarks in real-time with high-quality predictions. The pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face. The indexes of the 68 coordinates are visualized on the following Figure 1:

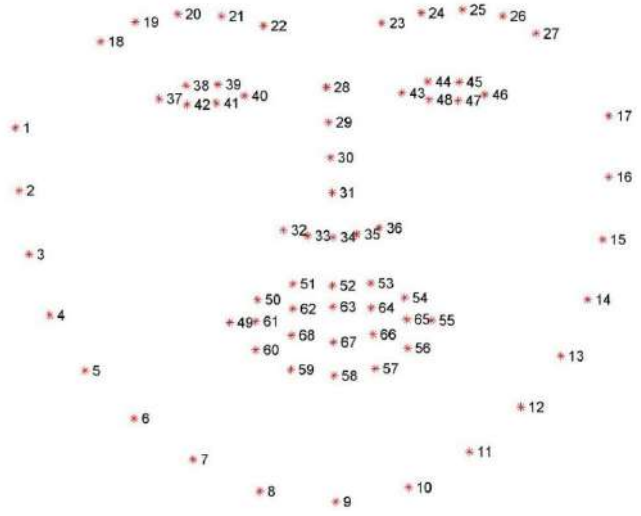
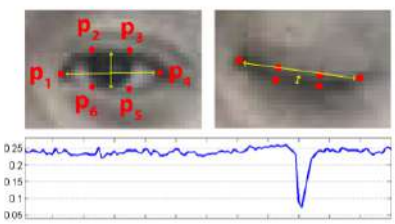


Figure 1: Indexes of the 68 coordinates of the facial landmarks.

The dlib facial landmark predictor was trained on the iBUG-300 W dataset[4], containing images and their corresponding 68 face landmark points. Other approaches of facial landmark detectors also exist, including for example a 194 point model that can be trained on the HELEN dataset. Nevertheless, regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data.
The implemented DSM module captures frontal facial images of the driver to assess fatigue levels based on the activity of his/her eyes. The driver's drowsiness is measured based on two metrics, the Eye Aspect Ratio (EAR)[5] and the PERcentage of Eye CLOSure (PERCLOS)[6]. To obtain the facial landmarks, we use the dlib toolkit, which provides us with 68 facial landmarks characterizing various facial features. From those 68 points, 12 points correspond to the eyes (6 landmarks for each eye). We use these landmarks to calculate the ratio of the vertical and horizontal lines defined by the eclipse that is fitted to the eye. This ratio is computed as:

[3] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1867-1874, doi: 10.1109/CVPR.2014.241.
[4] C. Sagonas, E. Antonakos, G, Tzimiropoulos, S. Zafeiriou, M. Pantic. 300 faces In-the-wild challenge: Database and results. Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild". 2016.
[5] F. You, X. Li, Y. Gong, H. Wang, and H. Li, "A real-time driving drowsiness detection algorithm with individual differences consideration" IEEE Access, vol 7, pp. 179396- 179408, 2019.
[6] D. F. Dinges and R. Grace, "PERCLOS: A valid psychophysiological measure of alertness as assessed by psychomotor vigilance" US Department of Transportation, Federal Highway Administration, Publication Number FHWA-MCRT-98-006, 1998



$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|},$$

Typical values indicating eyelid closure were determined at EAR < 0.2. In our tests, we found that an EAR threshold closer to 0.25 performs better in this simulation context. The PERCLOS measure is defined as the percent of the time the eyelid occludes the pupil (EAR < 0.25) within a K second moving window, where K can be tuned by the user. Typical values of K are around 60 seconds. Therefore, PERCLOS is calculated as:

$$PERCLOS\ (\%) = \frac{the\ sum\ of\ frames\ when\ eye\ is\ closed}{interval\ of\ frames}\ x\ 100\ \%$$

In other words, PERCLOS represents the percentage of the total frame that the eye is closed during a certain time interval. The eye is defined as closed based on predetermined PERCLOS values. The PERCLOS measure is calculated based on the area estimation of the iris, using the left and right iris area, and the total of both. There are three thresholds values (60%, 70%, and 80%) which are usually referred to as P60, P70, and P80, (e.g., P60 means the percentage of the total time that the eye is closed at least 60%.).
Alternatively, other metrics that can be used for the estimation of driver's drowsiness are presented below[7]:

- Inter-Event Duration (IED): blink duration measured from the point of maximum closing velocity to maximum opening velocity of the eyelid measured in seconds.
- Percent Time with Eyes Closed (%TEC): proportion of time eyes are closed, determined from the velocity of eyelid movement during eyelid closure.
- Blink Total Duration (BTD): duration of blinks measured in seconds from the start of closing to complete re-opening.
- Negative Amplitude-Velocity Ratio (–AVR): the ratio of the maximum amplitude to the maximum velocity of eyelid movement for the reopening phase of blinks.
- Positive Amplitude-Velocity Ratio (+AVR): the ratio of the maximum amplitude to the maximum velocity of eyelid movement for the closing phase of blinks.
- Percent Long Closures (%LC): proportion of time eyes are fully closed > 10 ms.
- Duration of Ocular Quiescence (DOQ): duration of no movements between the eyelid and ocular movement events, including blinks, saccades and smooth pursuit.
- Johns Drowsiness Scale (JDS): a composite measure of drowsiness calculated using weighted values of the other recorded ocular variables. JDS is calculated on a scale from zero to ten, with higher scores indicative of increased drowsiness

Pipeline of the algorithm

In this section, we will describe step by step the pipeline of the algorithm that we use.

- Start the video stream or a camera thread
  - Grab the frame from the threaded video file stream, resize it, and convert it to grayscale channels
- Detect faces in the grayscale frame

[7] Wilkinson VE, Jackson ML, Westlake J, Stevens B, Barnes M, Swann P, Rajaratnam SM, Howard ME. The accuracy of eyelid movement parameters for drowsiness detection. J Clin Sleep Med. 2013 Dec 15;9(12):1315-24. doi: 10.5664/jcsm.3278. PMID: 24340294; PMCID: PMC3836343.

- Initialize dlib's face detector (HOG-based) and create the facial landmark predictor
- Determine the facial landmarks for the face region

- Measurement's estimation
  - Grab the indexes of the facial landmarks for the left and right eye, respectively
  - Extract the left and right eye coordinates, and use the coordinates to compute the **eye aspect ratio** for both eyes
  - Define two constants, one for the eye aspect ratio (i.e., 0.25) to indicate blink and then a second constant for the number of consecutive frames (i.e., 48) the eye must be below the threshold for to set off the alarm
  - Compute the Euclidean distances between the two sets of vertical eye landmarks (x, y)-coordinates
  - Compute the Euclidean distance between the horizontal eye landmark (x, y)-coordinates
  - Compute the eye aspect ratio
  - Average the eye aspect ratio together for both eyes
  - Compute the convex hull for the left and right eye, then visualize each of the eyes
  - Check to see if the eye aspect ratio is below the blink threshold, and if so, increment the blink frame counter
  - Estimate PERCLOS

Functional diagram for detecting driver's drowsiness

Figure 2 illustrates in a graphical way the pipeline of the proposed DSM.

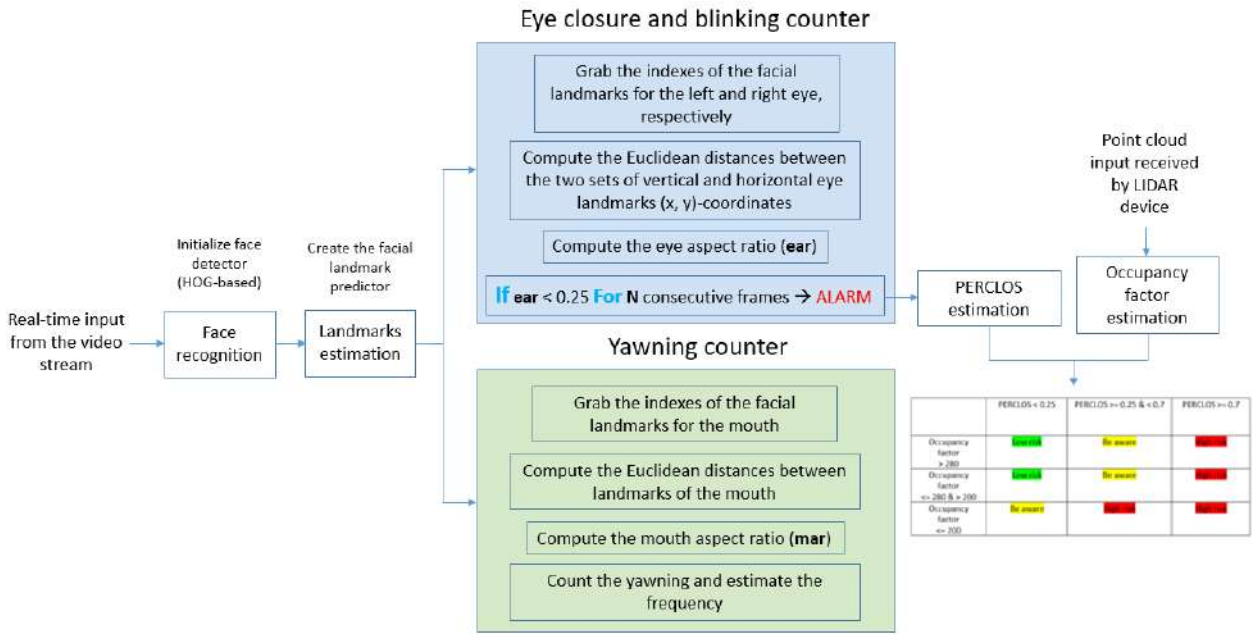


Figure 2: Functional diagram for the proposed DSM.

Figure 3 presents an example of the DSM approach while it is used in real-time implementation (right window) during a user drives in the Carla simulator under different traffic conditions.

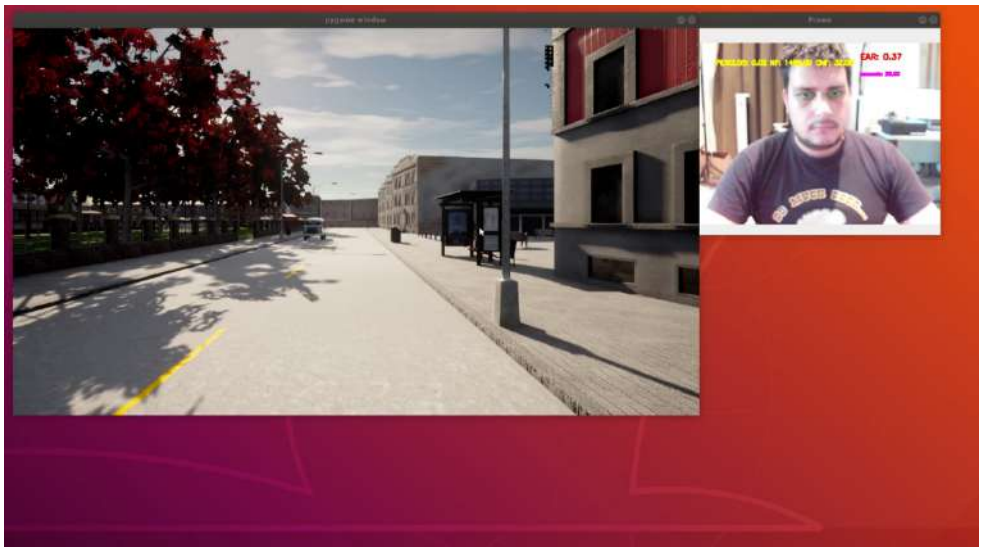


Figure 3: Example of the DSM approach while the user is driving in the CARLA simulator.

2.1.2 Android application

The estimations of the driver's drowsiness and attention levels are conducted utilizing Google's ML Kit, a standalone library which offers the capability of on-device machine learning processing. More specifically, the ML Kit provides the means for integrating machine learning capabilities into an application through the exposure of the so-called "vision APIs". Those APIs refer to both video and image analysis and can be exploited in a plethora of use cases, such as face detection, barcode scanning, pose detection, text recognition etc. Within the DSM application, the ML KIT was used for achieving a fast, efficient, and real-time face recognition and facial landmarks extraction.

2.1.2.1 Methodology

Since the application is aimed at providing continuous monitoring and assessment of the driver through the front camera of the smartphone, the core prerequisite is to place the smartphone in front of the driver with its front camera facing them. Figure 4 (left) illustrates the User Interface (UI) of the application, prompting the user to initiate the driving session, while Figure 4 (right) shows the set-up in the car.

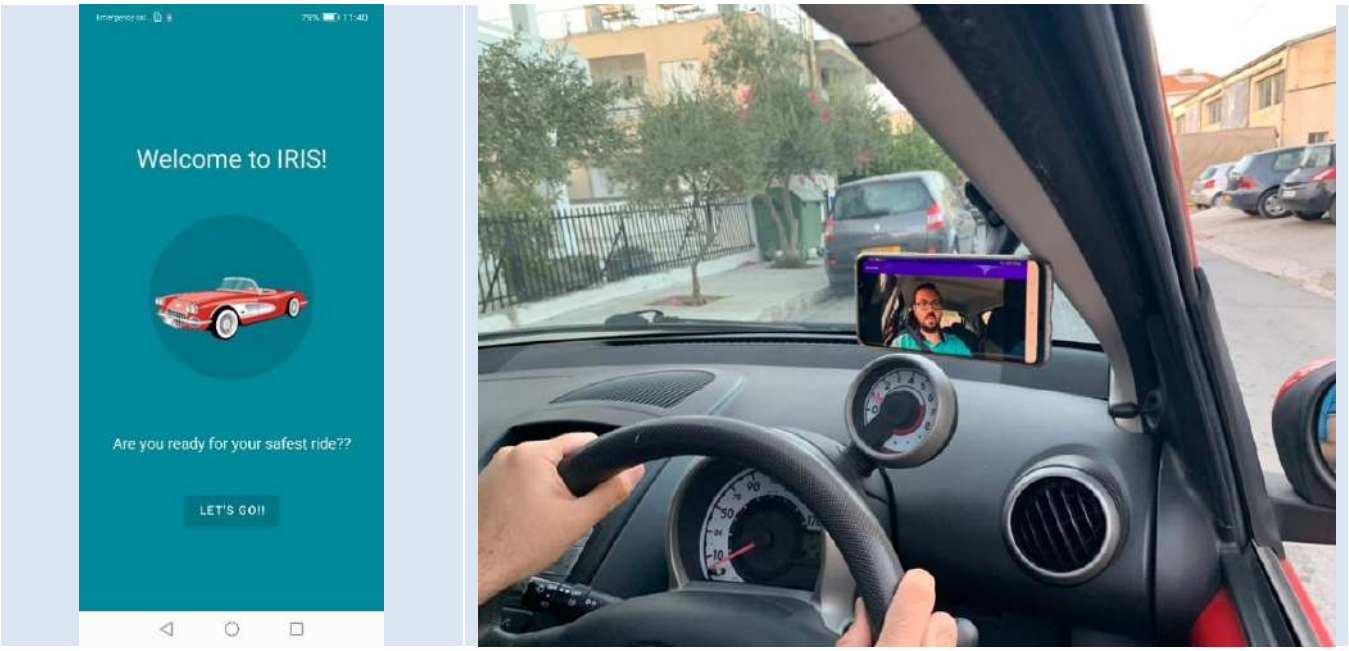


Figure 4: User Interface of the DSM application (left) and set-up on driver's car (right).

Once the driver starts the session, the front camera of the smartphone is engaged and starts capturing and analyzing the incoming frames. The analysis is conducted in a per-frame basis and the results obtained represent the detection and localization of the face within the frame and the extraction of the facial landmarks and their coordinates. As facial landmarks we consider points of interest within a face such as the eyes, cheeks, nose, lips and eyebrows. Figure 5 depicts the localization of the aforementioned points in a frame taken by the DSM application, both in conditions with and without sufficient lighting.

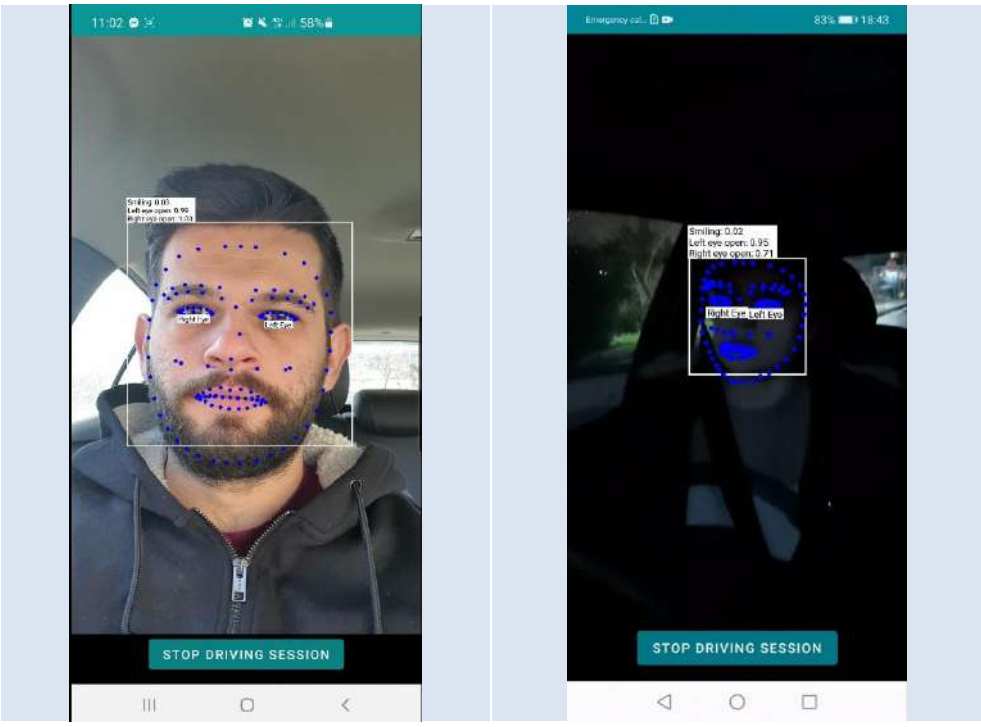


Figure 5: Localization of facial landmarks in sufficient light conditions (left); localization of facial landmarks in low light conditions (right).

Each incoming frame from the smartphone's camera is analyzed in order to determine i) if the driver has his/her eyes closed (either one or both of them), ii) if the driver is yawning, and iii) if the driver is looking left or right with respect to the camera. More specifically, the main functionalities of the DSM application are described below.

Eye tracking

The application is able to locate and track the driver's eyes and, by exploiting functionalities of the ML Kit library, to calculate the possibility of having his/her eyes closed. If the possibility is higher than a defined threshold for a number of consecutive frames, then a sound alert is raised in order to notify the driver.

Yawning

The application is able to define if the driver is yawning in a given frame by calculating the so-called Mouth Aspect Ratio (MAR). More specifically, The ML Kit's face detection results in a set of 38 points for the mouth as illustrated in Figure 6.

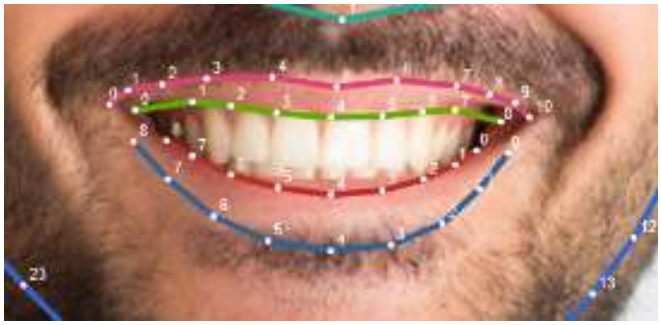


Figure 6: ML kit face detection – Point of Mouth.

From those points, we utilized points 3, 4 and 5, both from the bottom and the upper lip, alongside with points 0 and 10, which are the edges of the mouth. Having the coordinates of those points, we calculated MAR using the following equation, where the points refer to Figure 7.

$$\mathbf{MAR} = \frac{\|p_2 - p_8\| + \|p_3 - p_7\| + \|p_4 - p_6\|}{2\,\|p_1 - p_5\|}$$

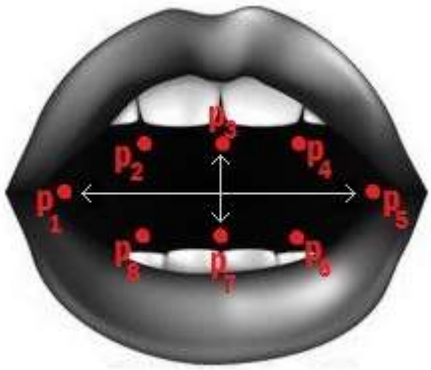


Figure 7: Mouth points utilized for calculating MAR.

If, for a number of frames within a specified timeframe, the driver is yawning, then the application throws a prompt message suggesting a break.

Head pose

The application is able to determine the orientation of the driver's face by utilizing information extracted from the ML Kit library regarding the angle of the face with respect to the camera. If the calculations show that the driver is looking right or left for a number of consecutive frames, then the application throws a prompt alert to notify the driver that he/she may be distracted.

JSON file generation

The application stores a JSON file, which includes all the aforementioned information per frame. This JSON file can be later used for further analysis, in order to acquire a comprehensive overview of the driver's behaviour during the driving session. A sample of the generated JSON file is given in Figure 8. A unique session ID and its timestamp are stored, along with unique frame IDs and their timestamps, the frame numbers, the number of detected faces in each frame, the driver's state and, finally, a flag whether the alert was fired or not. This information could be also potentially exploited in other third-party scenarios, like, e.g., in a forensics analysis if an accident occurs, in order to reconstruct the driving behaviour of the user.


```
[
  [
    "SessionUUID: c39997b9-80af-4dc2-b7fa-39dcd62fb8b7",
    "Session Timestamp: 2021-02-10T12:53:01.697Z"
  ],
  [
    "Frame Number: 2",
    "FrameUUID: 273b216a-d823-4fce-bade-b9012cd91760",
    "FrameTimeStamp: 2021-02-10T12:53:02.600Z",
    "Number of Detected Faces: 1",
    "Eyes Closed: false",
    "Yawning: false",
    "IsLookingLeft: false",
    "IsLookingRight: false",
    "Alert: false"
  ],
  [
    "Frame Number: 3",
    "FrameUUID: 6b8c4313-4d43-496b-887b-5aa799740d8f",
    "FrameTimeStamp: 2021-02-10T12:53:02.813Z",
    "Number of Detected Faces: 1",
    "Eyes Closed: false",
    "Yawning: false",
    "IsLookingLeft: false",
    "IsLookingRight: false",
    "Alert: false"
  ],
```


Figure 8: Generated JSON file.

2.1.2.2 Validation

The validation of the DSM application was conducted through a set of experiments in "real" situations (i.e., "on the road"), which were conducted both internally by CATALINK's team and by partner RTC. The requirement for those experiments was the existence of an Android-powered smartphone with the minimum OS version being v8.0 (Oreo). The users installed the DSM application and were asked to place the smartphone on their car's dashboard at a suitable position (i.e., either on the right or left side of the steering wheel), and to initiate a short driving session. The scenarios that the users were asked to experiment with were the following: i) keep their eyes closed for a few seconds, ii) yawn several times for a few minutes, and iii) look right or left for a few seconds. Furthermore, a driving session was also carried out during night, in order to test the application under insufficient lighting conditions.

Overall, the application managed to recognize signs of fatigue and distraction and notify the drivers on time via suitable alerts. On the other hand, both teams spotted a few issues, mainly regarding some false positives given in the yawning detection functionality and a delay in throwing alert messages when the driver is distracted. Partner CATALINK

CATALINK is currently working on improving the application based on the received feedback and further testing is scheduled in the coming weeks in order to ensure that all issues are resolved.

2.1.3 DEST based monitoring of driver's drowsiness

An overview of the ERT algorithm and the methodology followed as well as the structure of the resulting application is presented in this section. The FPGA implementation of the DEST package offers a Hardware/Software Codesign paradigm since alternative implementations (with different speed, accuracy, power consumption features) of the computationally intensive functions can be considered during the design of an application. Therefore, the DSM module and the alternative shape alignment models that can be used, will be described in detail in Task 4.1. Moreover, the details of the FPGA implementation of this module and its dynamic reconfiguration supported by the Xilinx XRT will be described in Task 5.1.

2.1.3.1 Methodology

The face alignment DSM application starts by opening a video stream (e.g., from a camera or a video) and periodically searches the input frames to detect a face using OpenCV library (Figure 9). All the frames could have been analyzed but that would lead to lower processing speed (frame rate) because the OpenCV face detection utilities are slow and it is assumed that the face cannot have changed significantly its position between 2 or 5 frames. The bounding box of the detected face is passed as an argument to the Tracker::Predict() function that implements the ERT algorithm.

In the ERT-based shape alignment approach, a mean shape of landmarks is continuously corrected in various cascade stages (e.g., 10) that are executed in the Tracker::Predict() routine. In each cascade stage a large number of regressor trees are visited in the Regressor::Predict() routine shown in Figure 9. In each regressor tree node the gray scale intensity of a pair of reference pixels around specific landmarks are compared to decide the next tree node that has to be selected starting from the root down to the leaves, in order to determine the appropriate factors for the correction of the landmark positions. The optimal hardware implementation of these predict() routines is one of the subjects of Task 5.1 where the dynamic reconfiguration will also be investigated with means offered by the Xilinx XRT. Different models can be generated by modifying parameters such as the number of cascade stages, the number of trees and reference pixels that are compared in these trees, the number of nodes and the depth of these trees, etc. These models have different accuracy, latency and require different number of resources for hardware implementations. They may also be suitable for different cases (male/female driver, driving in the night or in the daylight, etc). The selection of the appropriate combination of model and hardware implementation in a Hardware/Software Codesign environment, is the subject of the work of UoP in the Task 4.1.

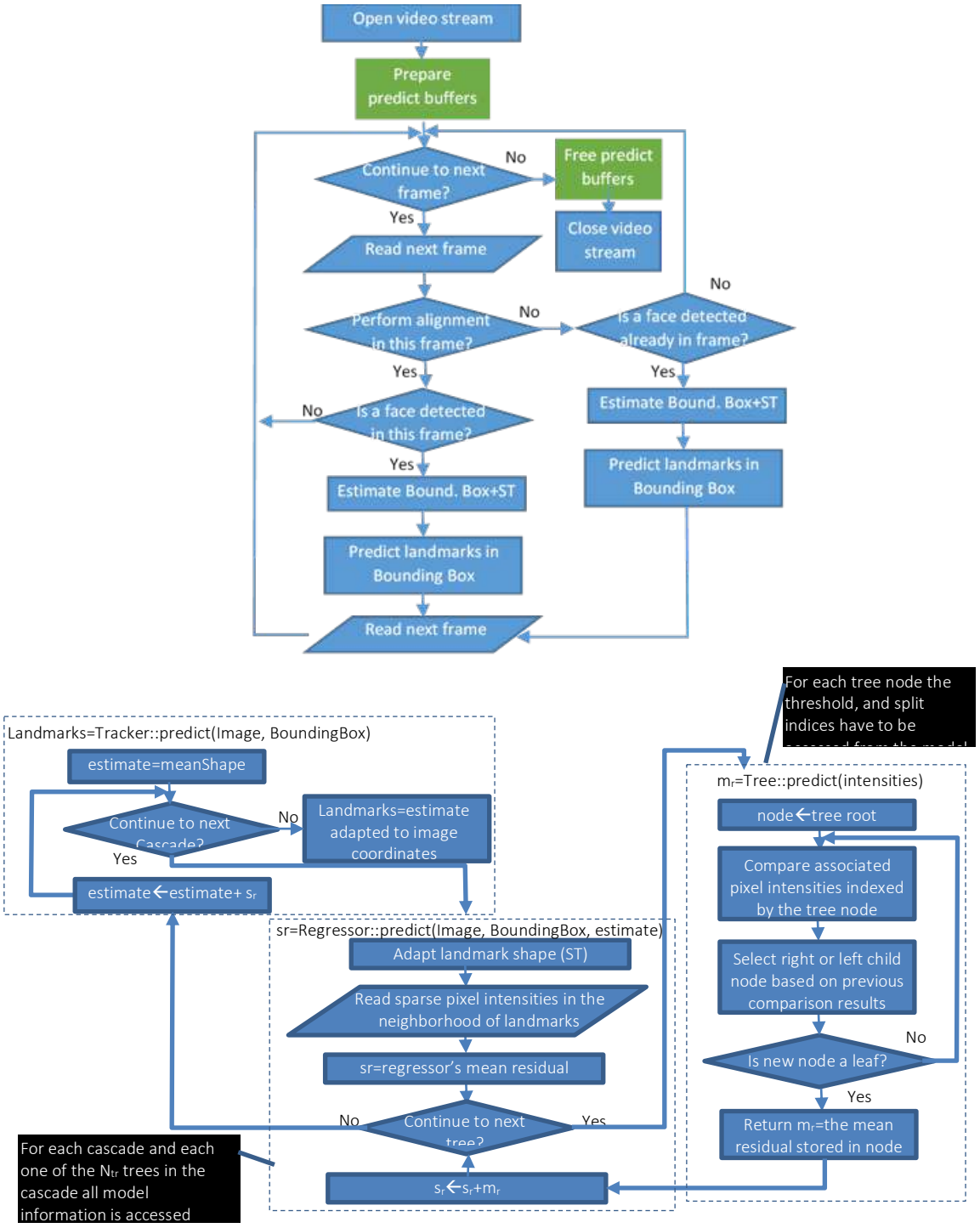


Figure 9: The functionality of the Predict() function.

2.1.3.2 Use and testing of the FPGA DSM module

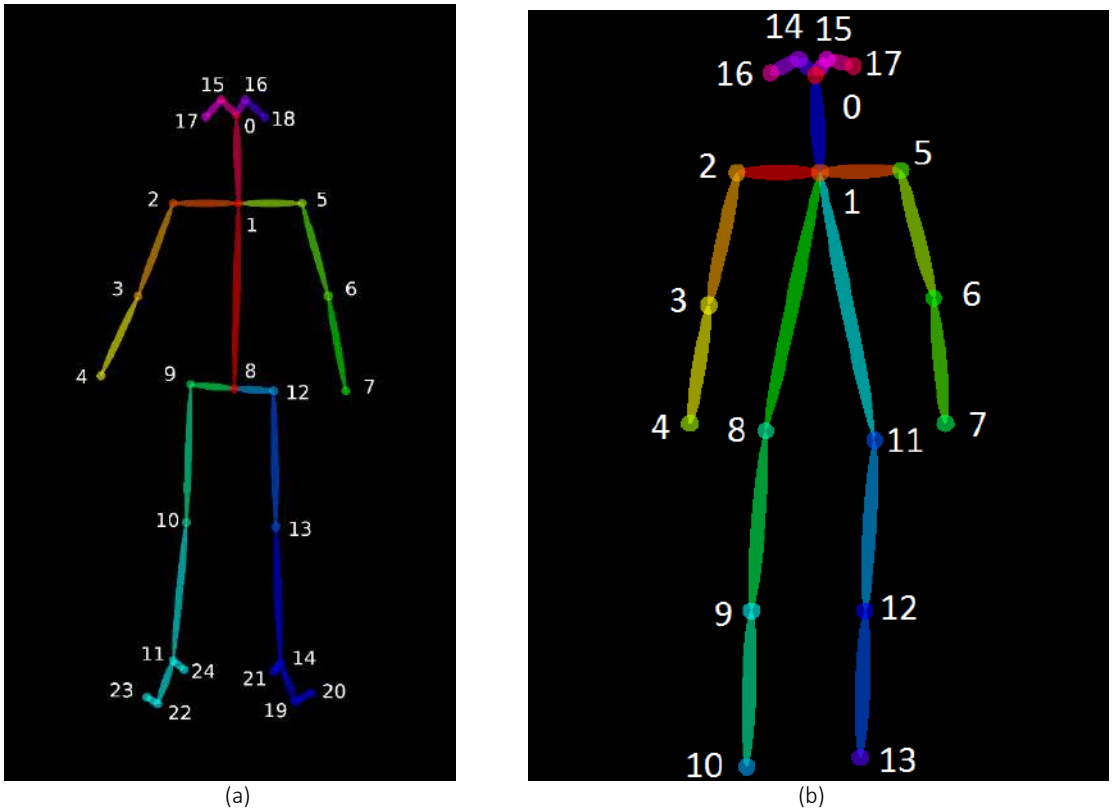
The FPGA DSM module developed by UoP will be installed in a car and tested using various environmental conditions and drivers. The camera will be placed on the dash of the car. The FPGA DSM application will detect yawning and sleepy eye blinks of the driver from the position of the eye and mouth landmarks as they are estimated in each frame. The position of the head can also be estimated to detect driver distraction. The results of this analysis, e.g., the estimated yawinings and eye blinks per time unit, will be displayed in a console and will be available to a higher-level application

that can alert the driver if drowsiness is detected. Alternatively, the input can also be a video stored in the SD-card of the FPGA to facilitate the appropriate testing of the application with various inputs. For debug reasons, an output video with the annotated landmarks and messages displaying the value of the various parameters measured is also stored in the SD-card of the FPGA.

2.2 Operator state monitoring

Human's pose estimation still remains an open problem in the research community. The initial efforts of the researchers were focused only on face landmarks detection. Next, the problem evolved into single and multi-person human pose estimation in the wild, including the whole body and foot landmarks. Several applications can immediately take advantage of the pose's landmarks detection. In general, almost any method that uses body information could also benefit from the face, hand, and foot detection, such as person re-identification, tracking, or action recognition.

Openpose is an algorithm that provides a whole-body pose estimation, following a multi-stage approach. It first obtains all body poses from an input image in a bottom-up fashion and then runs additional face and hand landmarks detectors for each detected person. Openpose detects human body, hand, facial, and foot landmarks (in total 135 landmarks) on single images. Nevertheless, in the applications that we search, the landmarks, representing the human body, are enough. To mention here that the number of landmarks that represents the human pose is not unique, and it depends on the selected (by the user) format. The most commonly used formats are the BODY_25 (25 landmarks) and the COCO (18 landmarks) as presented in Figure 10.



(a) (b)

Figure 10: (a) Pose Output Format (BODY_25), (b) Pose Output Format (COCO).

CNN architecture

The architecture of the used CNN is presented in Figure 11.

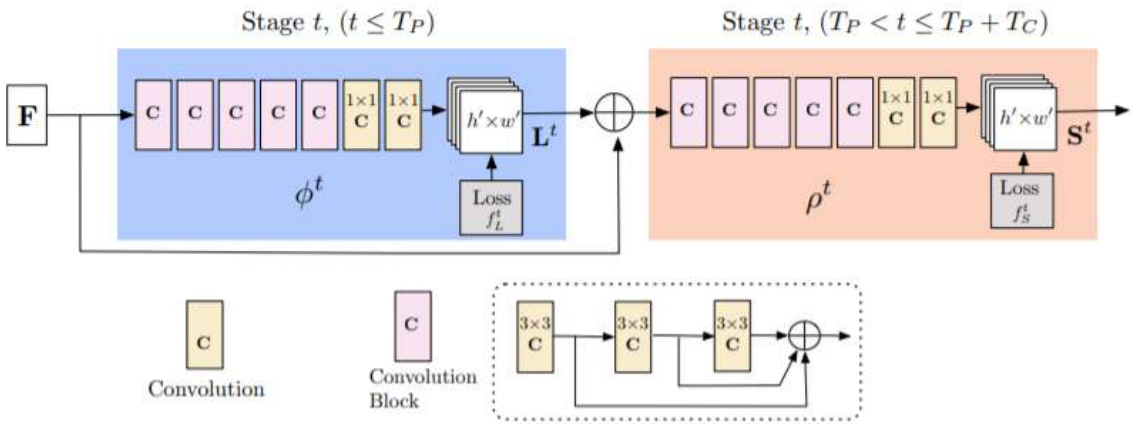


Figure 11: Architecture of the multi-stage CNN.

Features of the algorithm

- It has been trained in the COCO dataset of images
- It can combine pose, face and hand estimation of landmarks
- It can be used (off-line) in pre-recorded video or real-time using a camera
- It provides JSON files output with the coordinates of the landmarks and a confidence rate that can be used for the evaluation
- It runs on Images, Video, or Webcam
- It can provide different outputs (JSON, Images, Video, UI)
- The computational efficiency of the algorithm can be increased using a tracking approach, instead of pose estimation, for some frames

Assumptions for the implementation

- Human actors perform tasks in a manufacturing environment
- There are cameras, in static locations, that continuously monitor the operator's actions

Openpose implementation can be used for human-in-the-loop scenarios. In Figure 12, we present an example of the algorithm's implementation for the pose estimation of the operator to a video provided by the CRF's simulator showing some typical tasks and actions of the operator in the industrial environment. To mention here that the pre-trained dataset that is used, has been trained used images from real people, not VR simulated avatars.

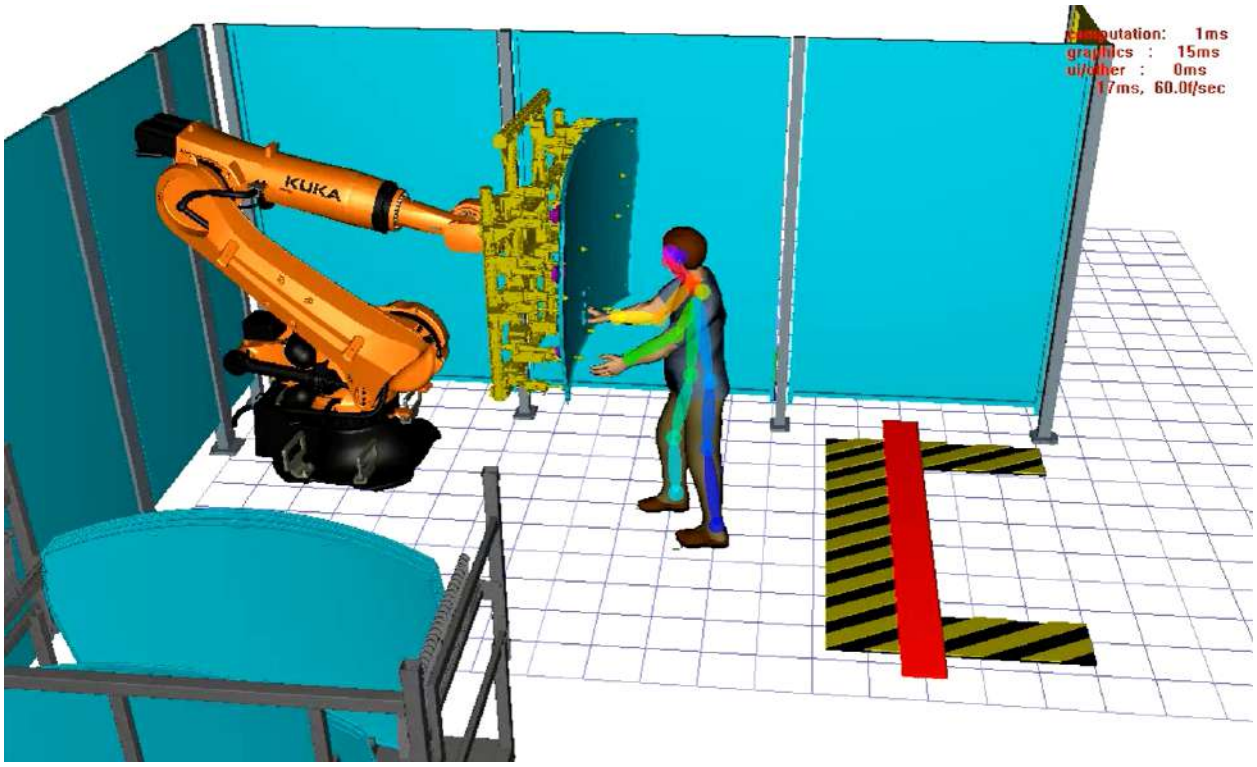
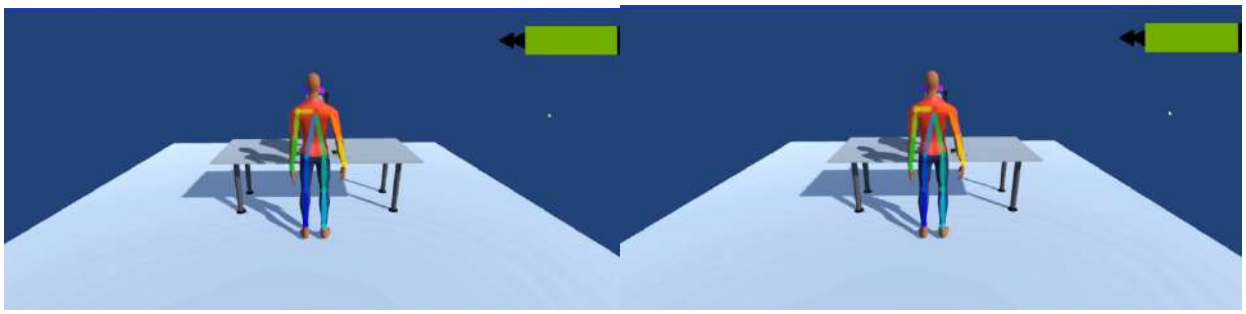


Figure 12: Estimation of operator's pose applied to the CRF's video.

The performance of the algorithm is evaluated under different parameterization such as the resolution of the input video as well as if the algorithm will use a tracking approach for the estimation (predicted approach less accurate) of the landmarks or it will calculate their position (more accurate). Table 1 - Table 2 present the mean confidence rates and fps of the processing for the different scenarios while Figure 13 - Figure 14 present the corresponding visual results.

Table 1: Evaluation of the openpose performance in high resolution videos.

High resolution videos (1920 x 956)		
Without tracking	Confidence ratio	0.953261
	fps	5
With 5 frames tracking	Confidence ratio	0.887867
	fps	13

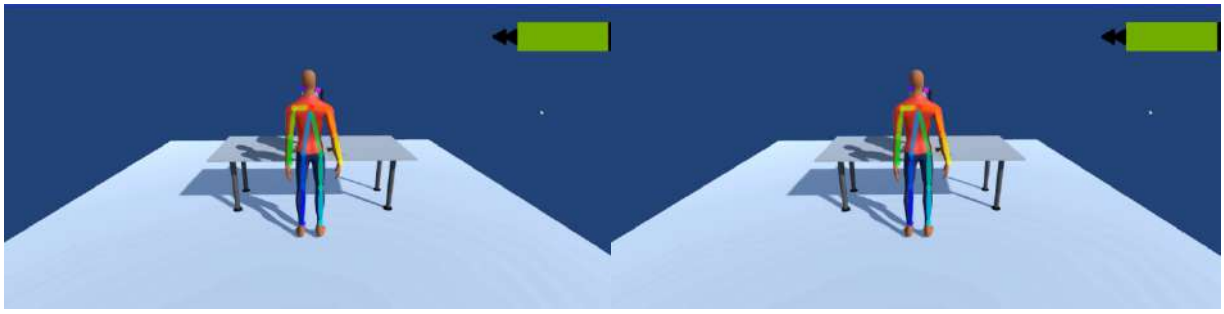


(a) (b)

Figure 13: Visual results of pose estimation on high resolution video into two cases, (a) without tracking, (b) with 5 frames tracking.

Table 2: Evaluation of the openpose performance in low resolution videos.

low resolution videos (640 x 480)		
Without tracking	Confidence ratio	0.914998
	fps	25
With 5 frames tracking	Confidence ratio	0.849164
	fps	30



(a) (b)

Figure 14: Visual results of pose estimation on high resolution video into two cases, (a) without tracking, (b) with 5 frames tracking.

Assuming that there are some static cameras in different locations of the manufacturing environment, our goal is to develop an algorithm that automatically selects this camera with the best view (i.e., back, front, side) based on the confidence value received by the pose estimation algorithm's output. In the following Figure 15 - Figure 17 , we show some screenshots of the 3 different views.

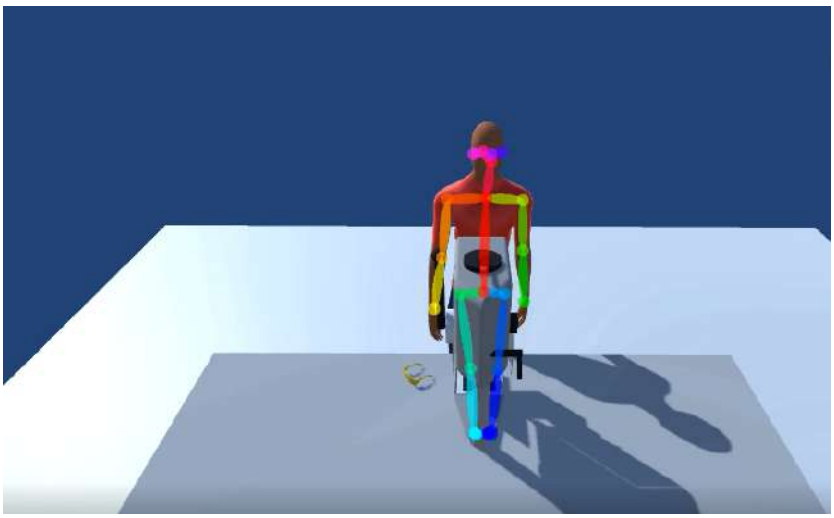


Figure 15: View from a camera in front of the operator (camera 1).

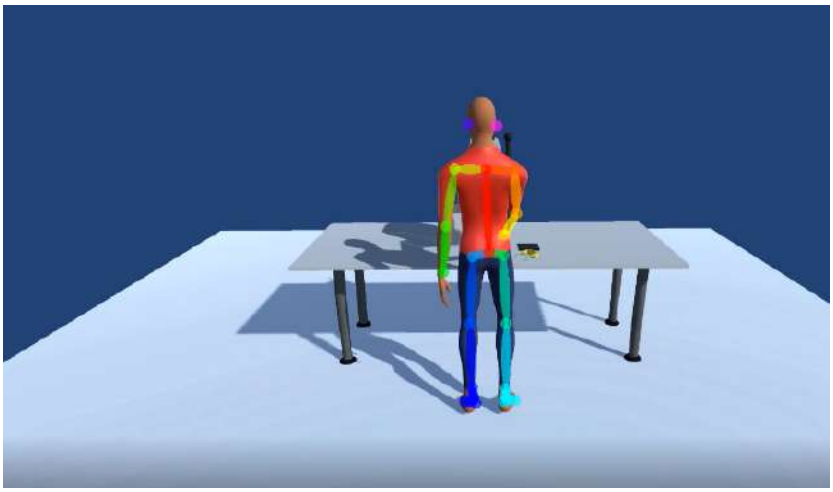


Figure 16: View from a camera behind the operator (camera 2).

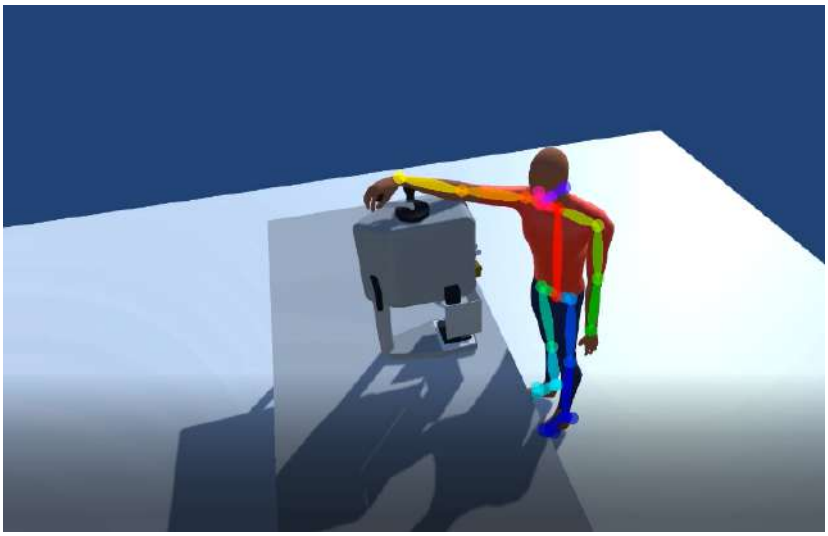


Figure 17: View from a camera on side the operator (camera 3).

Figure 18 presents the selected camera (camera 2 in this example) which is based on the lowest mean confidence rate, estimated by all presented landmarks, of the three cameras.

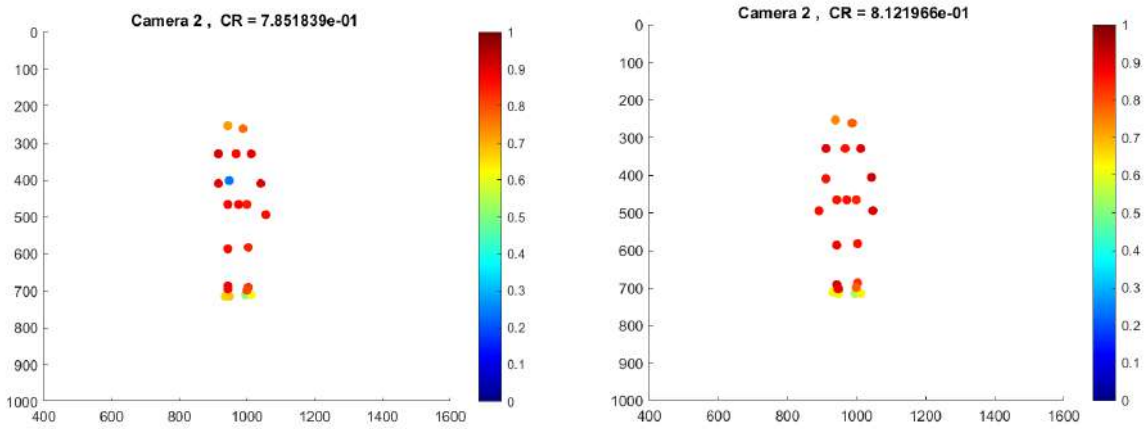


Figure 18: Selected camera, based on the mean confidence rate of all landmarks and heatmap showing the confidence rate of each landmark.

Additionally, in some tasks, we need to give more emphasis to specific areas of the operator's body since the main action takes place there. In Figure 19, we present the algorithm's result when specific points of interest are selected (e.g., above the torso), related to the operator's task that we want to monitor and analyze.

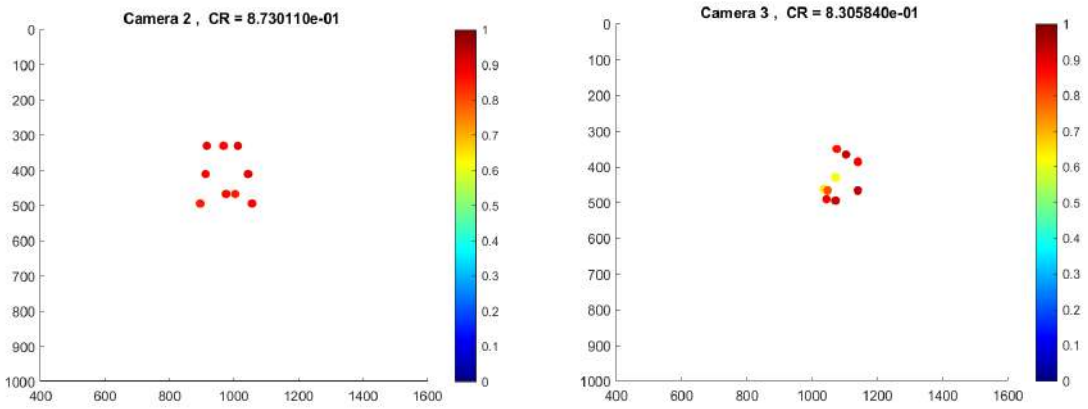


Figure 19: Selected camera, based on the mean confidence rate of the preferable landmarks and heatmap showing the confidence rate of these landmarks.

Evaluation of the landmark's prediction based on the ground truth

The pose prediction algorithm returns the 2D coordinates of the landmark points and a confidence rate showing how confident the network is about this prediction. We estimate the landmarks of the user's pose, and we attend to evaluate the accuracy of this estimation. For the evaluation, we use the 3D ground-truth landmarks that have been manually set to the 3D model via Unity's environment. Then, for each frame, we project these points to the 2D coordinate system of the image in order to estimate the MSE metric between ground truth and estimated 2D points that have been extracted as a JSON file.

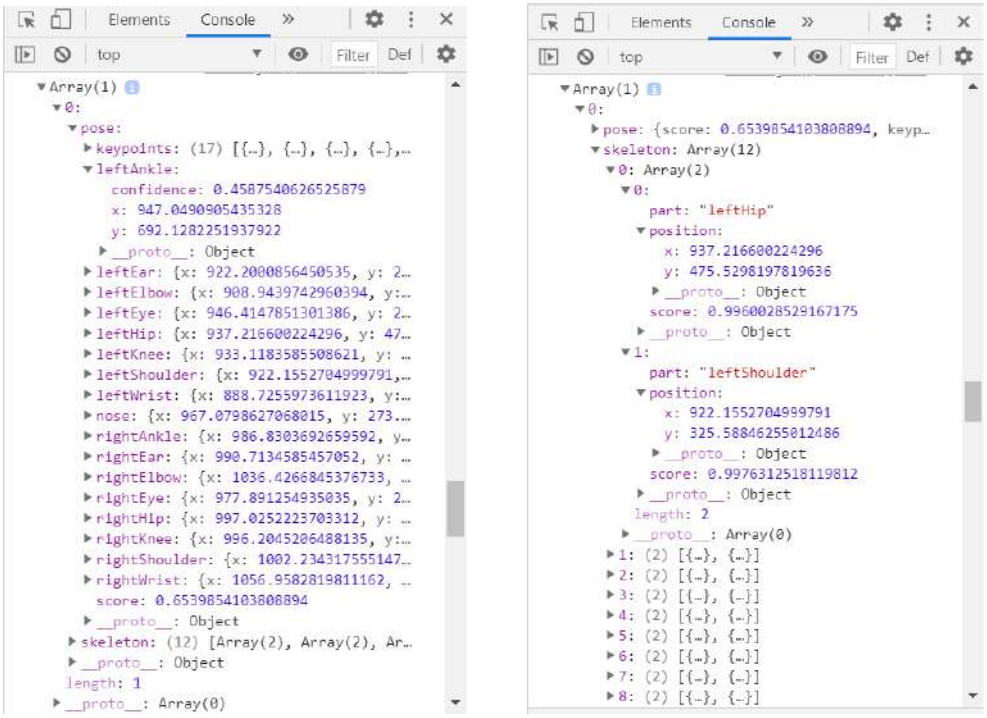


Figure 20: Example of the format of the landmarks' 2D coordinates and their corresponding confidence rate.

More specifically, for the evaluation of the pose recognition output (i.e., pose landmarks) with the ground truth landmarks of the 3D avatar model which have been pre-defined via the simulator, we follow the next steps.

1) Extract 2D coordinates of markers from 3D space in Unity
2) The 3D ground truth landmarks have been manually set to the 3D model via Unity's environment.

- Initiate unity camera to look at our DHM and compute its View (world space -> camera space)
  + Projection Matrix (points are normalized in a 2D coordinates view between [-1,1] clip view/space)
- List with 3D objects (markers) passes through a function that calculates their screen position.
- Viewport space coordinates ([0,1]) of the 3D markers are calculated.
- Screen space coordinates (pixel values) are calculated with x ranging [0, 1920] and y [0, 984], since camera resolution is (1920, 984)
- Last but not least the origin that starts counting the pixels had to change to the upper left in order to match with Openpose's coordinates system.

Figure 21 depicts the MSE per each frame of the video and additionally we have highlighted some values presenting also the corresponding confidence value of the openpose's output. We can observe that lower values of confidence (<0.96) values correspond to higher values of MSE values.

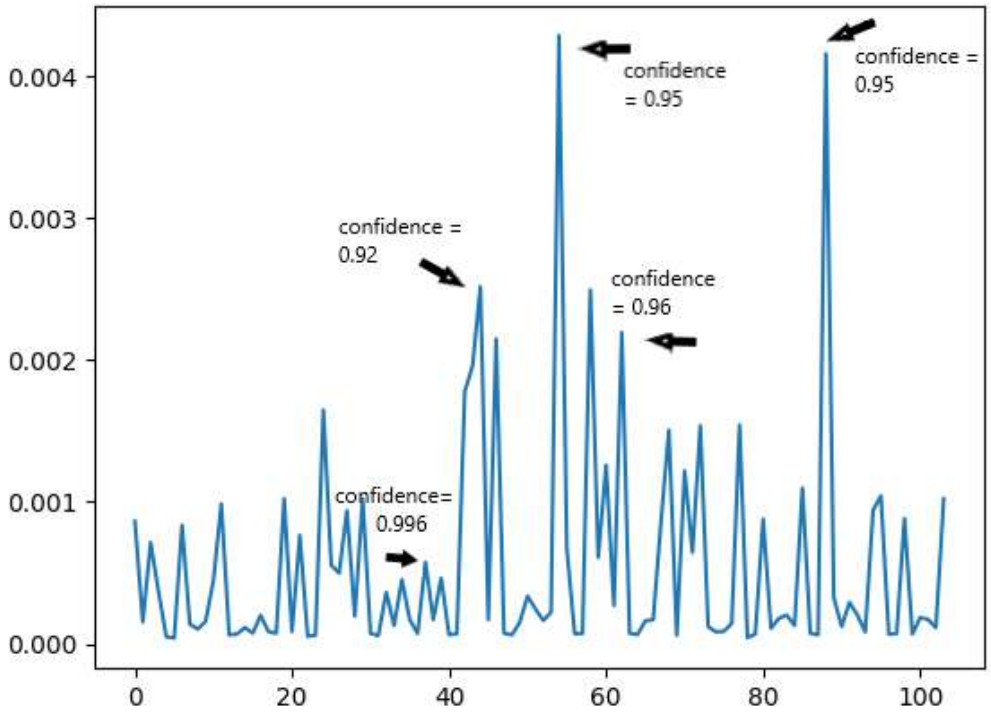


Figure 21: MSE of the estimated and the ground truth landmarks per each frame.

3 Multi-modal scene understanding, localization and mapping

3.1 Multi-modal scene understanding

3.1.1 Image driven scene understanding

Object detection has evolved considerably since the appearance of deep convolutional neural networks. Nowadays, there are two main branches of proposed techniques. In the first one, the object detectors, using two stages, generate region proposals which are subsequently classified in the categories that are determined by the application at hand

(e.g., vehicles, cyclists and pedestrians, in the case of autonomous driving). Some important, representative, high performance examples of this first branch are Faster R-CNN[8], Region-based Fully Convolutional Network (R-FCN)[9], Feature Pyramid Network (FPN)[10] and Mask R-CNN[11]. In the second branch, object detection is cast to a single-stage, regression-like task with the aim to provide directly both the locations and the categories of the detected objects. Notable examples here, are Single Shot MultiBox Detector (SSD)[12], SqueezeDet[13], YOLOv3[14] and EfficientDet[15].

Although two-stage detectors demonstrate better performance than the single-stage counterparts, the latter have lower computational and storage requirements which leads, generally, to faster inference time. In autonomous driving, Advanced Driver Assistance Systems (ADAS) rely on embedded systems with limited resources. ADAS is responsible for executing various machine learning tasks, including object detection, meaning that efficient implementations that take into account those limitations are critical. To this end, single-stage detectors have been particularly studied for autonomous driving by either proposing specialized, compact deep models (e.g., SA-YOLOv3[16], Mini-YOLOv3[17]) or applying MCA techniques[18] to existing, pre-trained models.

Two deep detection network architectures, namely SqueezeDet and Resnet50ConvDet, were employed for the evaluation of the presented weight sharing approach. They are fully convolutional detection networks presented by Wu et al.[19], consisting of a feature-extraction part that extracts high dimensional feature maps for the input image, and ConvDet, a convolutional layer to locate objects and predict their class. For the derivation of the final detection, the output is filtered based on a confidence index also extracted by the ConvDet layer. Figure 22 presents the overall architecture of the deep networks, the convolutional volume kernel shapes and the feature tensor shapes.

As it can be observed from Figure 22, the feature-extraction (convolutional) part of SqueezeDet is based on SqueezeNet which is a fully convolutional neural network that employs a special architecture that drastically reduces its size while still remaining within the state-of-the-art performance territory. Its building block is the "fire" module that consists of a "squeeze" 1×1 convolutional layer with the purpose of reducing the number of input channels, followed by 1×1 and 3×3 "expand" convolutional layers that are connected in parallel to the "squeezed" output. SqueezeNet consists of 8

[8] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, 2017.

[9] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: Object detection via region-based fully convolutional networks,” Adv. Neural Inf. Process. Syst., pp. 379–387, 2016.

[10] Lin, Tsung-Yi, et al. "Feature pyramid networks for object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[11] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 2, pp. 386–397, 2020.

[12] W. Liu et al., “SSD: Single shot multibox detector,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9905 LNCS, pp. 21–37, 2016.

[13] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving,” IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work., vol. 2017-July, pp. 446–454, 2017.

[14] J. Redmon and A. Farhadi, “YOLO v3,” Tech Rep., pp. 1–6, 2018.

[15] M. Tan, R. Pang, and Q. V. Le, “EfficientDet: Scalable and efficient object detection,” Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., pp. 10778–10787, 2020.

[16] D. Tian et al., “SA-YOLOv3: An Efficient and Accurate Object Detector Using Self-Attention Mechanism for Autonomous Driving,” IEEE Trans. Intell. Transp. Syst., pp. 1–12, 2020.

[17] Q. C. Mao, H. M. Sun, Y. B. Liu, and R. S. Jia, “Mini-YOLOv3: Real-Time Object Detector for Embedded Applications,” IEEE Access, vol. 7, pp. 133529–133538, 2019.

[18] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey,” Proc. IEEE, vol. 108, no. 4, pp. 485–532, 2020.

[19] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving,” IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work., vol. 2017-July, pp. 446–454, 2017.

such modules connected in series.

On the other hand, the backbone of ResNetDet, presented in Figure 23, is based on the convolutional layers of ResNet50[20], whose building block consists of three layers, stacked one over the other. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions. The convolutional part of ResNetDet consists of 13 such blocks.

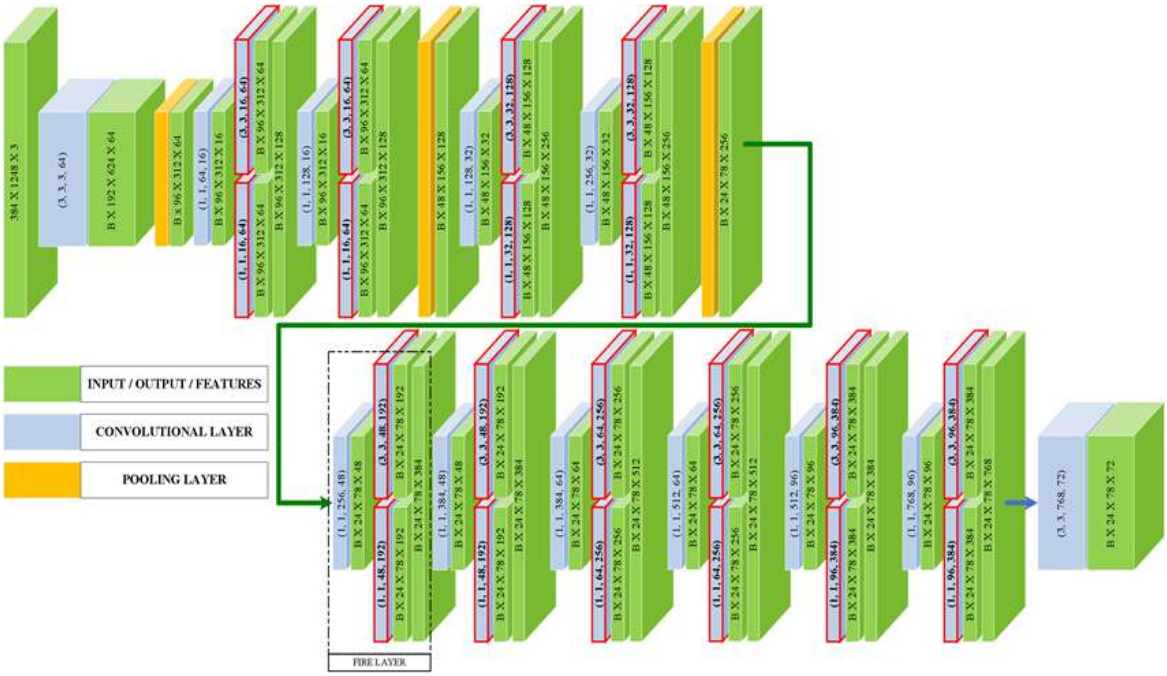


Figure 22: SqueezeDet architecture.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, vol. 19, no. 2, pp. 770–778.

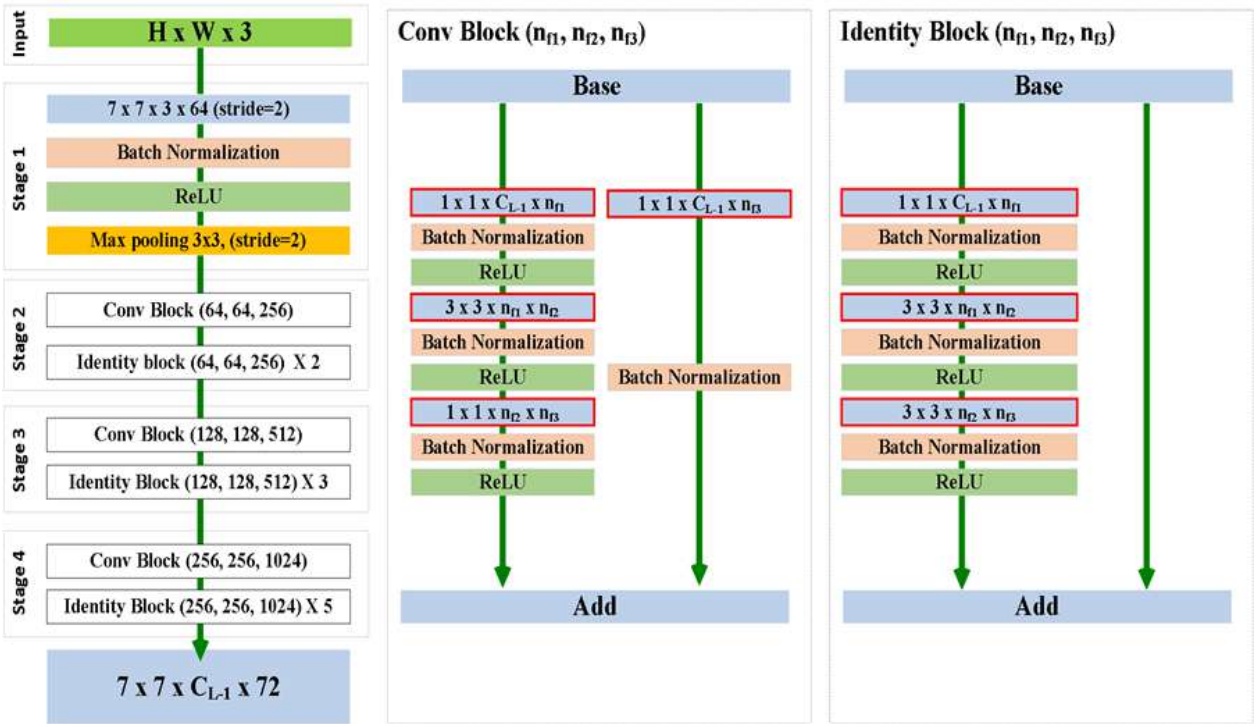
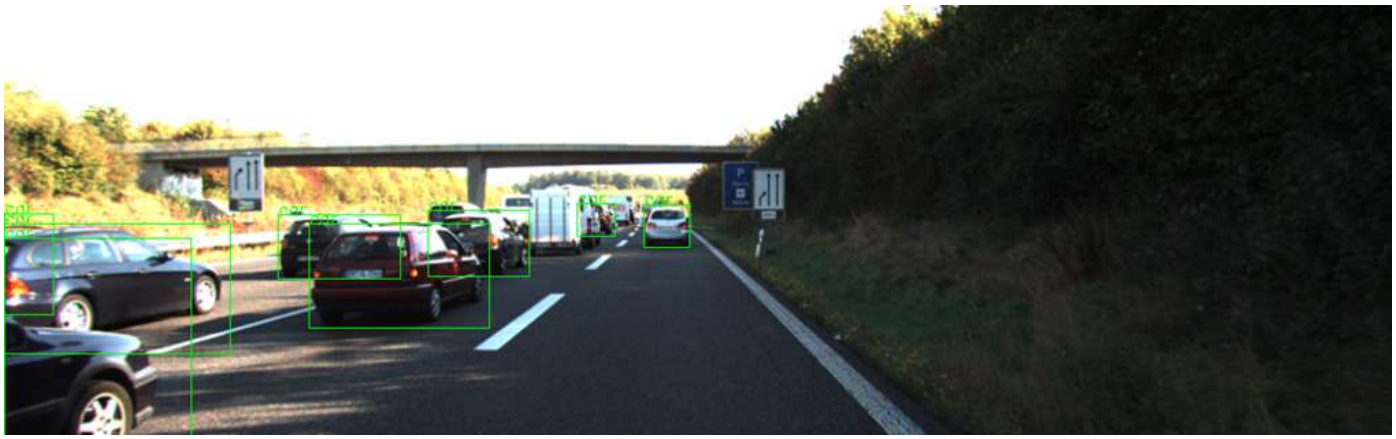


Figure 23: Resnet50convdet architecture.

3.1.1.1 KITTI Dataset

The KITTI odometry dataset [21] has been used for training the DNN models under consideration. In total, there are 7477 images depicting traffic scenes (indicative examples shown in Figure 24), of which $N_{tr} = 5980$ and $N_{val} = 1497$ are used for training and validation purposes, respectively. The objects are labeled using the three categories of cars, cyclists, and pedestrians. In our study, three classes are mainly examined, cyclists, pedestrians and cars annotated with bounding boxes containing the objects in the 3D scene. Furthermore, 3716 annotated Velodyne point cloud scenes were used for training and 3769 annotated Velodyne point cloud scenes were used for testing and validation.



[21] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 3354–3361.

Figure 24: KITTI dataset examples.

3.1.1.2 Simulation setup

Both networks were trained with the KITTI odometry dataset consisting of 7477 color traffic scenes images of pixels. Three classes are taken into account, namely, cyclists, pedestrians and cars which were manually annotated with bounding boxes containing the objects in the scene. A significant observation regarding the dataset is that not all objects of the same class are labeled in each and every image. Such a fact plays a role in the evaluation of the detection outcome as our analysis will reveal. The dataset was split in for training and validation, respectively, resulting in training examples and validation examples.

Moreover, a data augmentation scheme was adopted, according to which the bounding boxes drift by $k_x * 150$ and $k_y * 150$ pixels across the x-axis and the y-axis, respectively, where $k_x, k_y \sim U(0,1)$. A 50% probability is also assumed to flip an object. For the training of the SqueezeDet architecture, Stochastic Gradient Descent (SGD) was employed with the following values for the hyperparameters (determined via experimentation); batch size $B = 8$, learning rate $LR = 10^{-4}$, with a weight decay rate $D_W = 10^{-4}$, a learning rate decay rate of $D_{LR} = 2 * LR/N_e$, number of steps $N_s = 3 \times N_{tr}$ and a dropout rate of 50%, over a total of $N_e = 300$ epochs. Training and testing took place in an NVIDIA GeForce GTX 1080 graphics card with 8GB VRAM and compute capability 6.1 in a Intel(R) Core(TM) i7-4790 CPU @ 3.60Hz based system with 32GB of RAM.

Likewise, for Resnet50ConvDet, we also employed SGD with hyperparameter values as in the case of SqueezeDet. Training and evaluation of Resnet50ConvDet took place in an NVIDIA GeForce RTX 2080 with 16GB VRAM and compute capability in a Intel(R) Core (TM) i7-4790 CPU @ 3.60Hz based system with 16GB of RAM.

In all cases, training took place with a data augmentation scheme where the bounding boxes drift by pixels across the

x-axis and pixels across the y-axis, where. A probability is also assumed to flip the object.

For each detection, the Intersection Over Union (IOU) score is computed as the ratio of area of intersection to the area of union between the predicted and ground-truth bounding boxes. A true positive occurs when IOU and the predicted class is the same as the ground-truth class. A false positive occurs when IOU or a different class is detected, meaning that unmatched bounding boxes are taken as false positives for a given class. Precision, recall and mean average precision (mAP) are subsequently calculated[22].

Table 3: Average precision and recall score for SqueezeDet and ResNet50Convdet.

Class	Car		Cyclist		Pedestrian	
Score (%)	*AP*	*RC*	*AP*	*RC*	*AP*	*RC*
SqueezeDet	88.9	73.3	78.5	54	75.6	56.6
ResNet50Conv Det	94.3	83.9	78.6	66.2	77.7	66.8

3.1.2 LIDAR driven scene understanding

3D object detection from LIDAR point clouds is mainly a data-driven task due to the lack of apparent structure in the data. Deep fully convolutional networks have been traditionally employed in the literature since 2016, with the main evolutionary elements to concern i) the transformation of the 3D point cloud, ii) the network structure and iii) the utilization of feedback loops or abstraction layers for a multiscale feature extraction approach. Initial attempts [23] projected the 3D points in a 2D level and used traditional 2D fully convolutional networks reporting accuracy of 71.0% for moderate difficulty cars. The use of 3D fully convolutional networks increased the accuracy to 75.3% but also increased the computational complexity. Yan et al.[24] proposed sparse convolutional networks improving training and inference times and reaching a reported accuracy of 79.46% for moderate difficulty cars, according to KITTI benchmarks. Pointpillars[25] proposed a novel encoder that utilizes PointNets to learn a representation of point clouds organized in vertical columns (pillars) presenting an accuracy of 74.31% in the same category. 3D object detection with region proposals was introduced in PointRCNN of Shi et al.[26]. They used two stages, where the first stage generates 3D proposals and the second refines them reporting an accuracy of 75.64%. An extension of PointRCNN is the Part-A^2

[22] D. M. W. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," pp. 37–63, 2020.

[23] Li, Bo, Tianlei Zhang, and Tian Xia. "Vehicle detection from 3d lidar using fully convolutional network." arXiv preprint arXiv:1608.07916 (2016).

[24] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," Sensors (Switzerland), vol. 18, no. 10, pp. 1–17, 2018.

[25] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 2019-June, pp. 12689–12697, 2019.

[26] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 2019-June, pp. 770–779, 2019.

Net[27] encompassing a part-aware proposal part and an aggregation part reporting an of 78.49%. PV-RCNN[28] is another novel approach that combines 3D sparse convolutions in a voxelized space with region proposals and an abstraction layer reporting an average precision of 81.43%.

Here, the two object detection schemes that will be considered, namely, PointPillars and PV-RCNN, are briefly presented. The PointPillars network introduces the notion of a Pillar. Based on those Pillars, this network removes the need for 3D convolutions, which have been central to networks like VoxelNet[29] and second, by utilizing striCatalinky 2D convolutions, thus, achieving both high precision and fast inference.

The architecture of PointPillars consists of three stages as depicted in Figure 25 (a). The first stage transforms the point cloud into a pseudo-image. By grouping the points of the cloud into vertical columns, called pillars, that are positioned based on a partition of the plane, this stage summarizes the information of the points per pillar into 1D vectors. These vectors are rearranged appropriately to construct the pseudo-image that will feed the next stage. The second stage consists of a feature extraction backbone network that provides a high-level representation. This representation is subsequently processed by the third stage which is the adopted object detector, producing 3D bounding boxes and confidence scores for the classes of interest. In terms of computational complexity, the backbone network of the second stage, consisting of a number of 2D convolutions and 2D transpose convolutions, requires more than of the involved operations and, thus, we will focus on this stage in the following for its acceleration.

The second object detector for point clouds that will be considered, is the recently proposed PV-RCNN[30]. On one hand, this system capitalizes on ideas from grid-based object detectors that transform the irregular point clouds into regular representations that can be processed efficiently by ordinary convolutional layers, limiting, however, their performance by the resolution of the adopted grid. On the other hand, PV-RCNN also exploits ideas from point-based object detection, which operates directly on the points of the cloud, removing the need for point cloud discretization with increased, however, computational complexity.

In more detail, the architecture of PV-RCNN is depicted Figure 25 (b). PV-RCNN, first, transforms the point cloud into voxels which are subsequently processed by a voxel backbone network consisting of 3D sparse convolutions. Based on its output, 3D region proposals are produced using the BEV backbone network whose structure is depicted in Figure 25 (b). PV-RCNN also samples a subset of the points, named key points, and associates summary features by appropriately concatenating information extracted by the key points themselves and corresponding information extracted from different layers of the voxel backbone network. The information of the 3D region proposals along with the key-point features are appropriately merged over equispaced grid points defined in each 3D region proposal. This enhanced information is eventually processed to procedure the improved final confidence scores for the classes and the 3D bounding boxes. In the following, we will focus on the BEV backbone for studying the impact on the performance of the PV-RCNN.

Both networks were trained with the KITTI 3D object detection benchmark consisting of 7481 training images and 7518 test images as well as the corresponding point clouds, comprising a total of 80.256 labelled objects In our study, three classes are mainly examined, cyclists, pedestrians and cars annotated with bounding boxes containing the objects in the 3D scene. 3716 annotated Velodyne point cloud scenes were used for training and 3769 annotated Velodyne point cloud scenes were used for testing and validation. For the deployment and retraining of PointPillars and PV-RCNN, the

[27] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network," IEEE Trans. Pattern Anal. Mach. Intell., pp. 1–1, 2020.

[28] Shi, Shaoshuai, et al. "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

[29] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., pp. 4490–4499, 2018.

[30] S. Shi et al., "PV-RCNN: Point-voxel feature set abstraction for 3D object detection," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., pp. 10526–10535, 2020.

OpenPCDet framework[31] was employed. For the initial evaluation, pre-trained instances were used, while for the retraining, the Adam optimizer was employed with learning rate $l_r = 0.003$, weight decay rate $D_W = 10^{-2}$ and a batch size $B = 4$. Training took place in an NVIDIA Geforce RTX 2080 with 16GB VRAM and compute capability 7.5. Furthermore, for the Pointpillars network the detection accuracy was evaluated on NVIDIA Jetson TX2, while for the PV-RCNN network, due to model size, the detection accuracy was evaluated on the NVIDIA Geforce RTX 2080. Table 4 presents average precision for PoinPillar and PV-RCNN for Car, Pedestian and Cyclist classes.

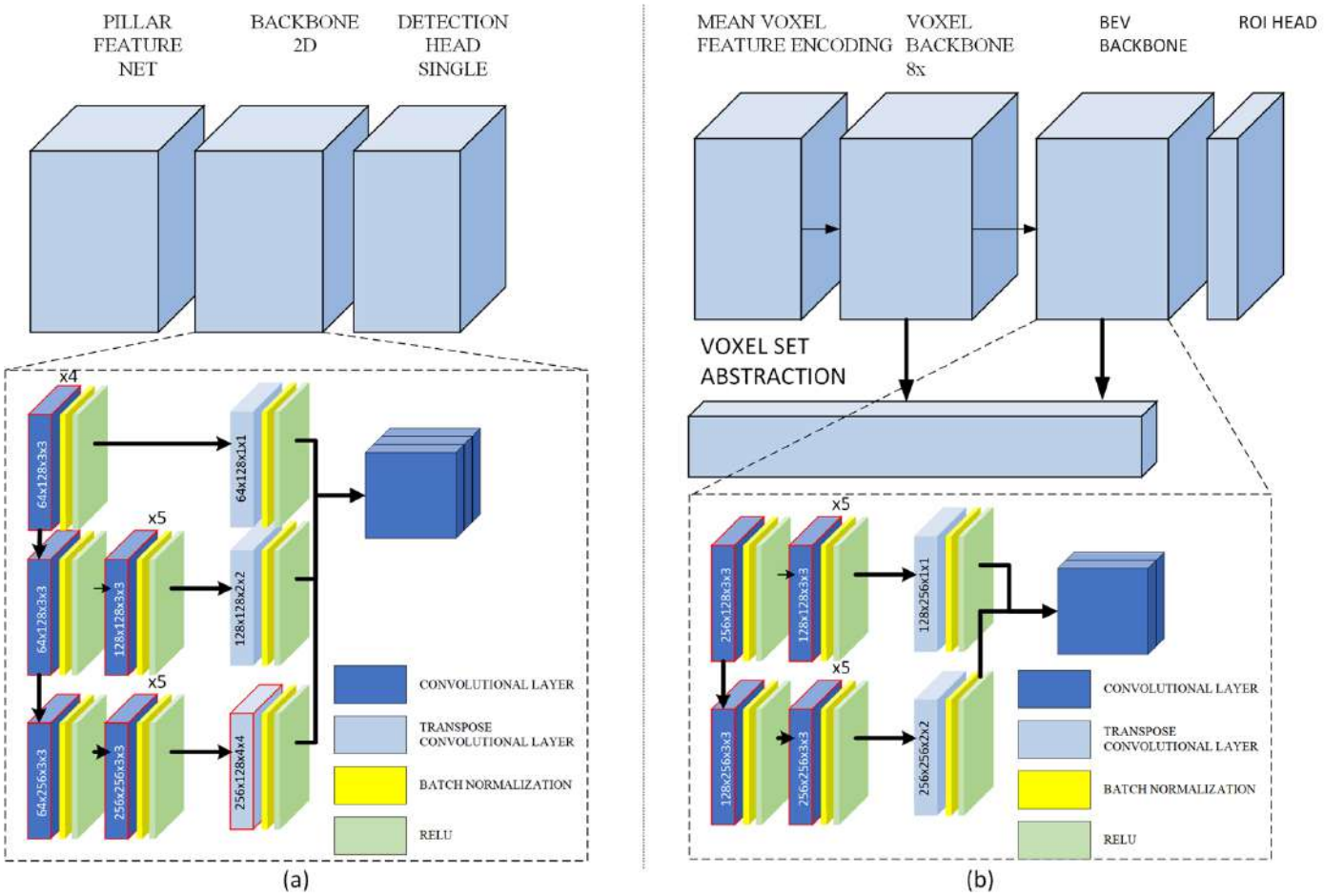


Figure 25: a) Pointpillar architecture b) PVRCNN architecture.

Table 4: Average precision for Point pillar and PVRCNN.

	Car			Pedestrian			Cyclist		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PointPIllar	92.04	88.06	86.66	61.60	56.01	52.05	85.26	66.24	62.22

[31] O. D. Team, "OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds." 2020.

PV-RCNN	94.51	90.19	88.09	71.12	64.12	59.89	91.35	74.63	69.89

3.1.3 Multimodal fusion

There is a variety of strategies aiming to increase the overall performance, by fusing multiple modalities. We can divide the previous types into three categories: Early fusion, Late fusion and Deep fusion. Taking into consideration the first category, modalities are combined at the beginning of the process and extract the shared information on data, by jointly processing the raw data measurements acquired by different sensors. Late fusion performs the synthesis of valuable information at the final stage of feature extraction, where fusion occurs. Finally, a more general fusion scheme refers to Deep fusion. By exploiting the capability of DL to discover high-level data representation, the Deep fusion can effectively find the joint data representation by combining the features extracted at the intermediate layers of deep neural networks. In our case, late fusion has been applied.

3.1.3.1 Point Projection

For projecting lidar points to the image plane, we are using the calibration data provided by the KITTI benchmark. lidar data have been captured by a Velodyne HDL-64E S2 sensor and each point is stored with its (x, y, z) coordinate and an additional reflectance value (r). The number of points per scan on average for each file/frame is $\sim 120{,}000$ 3D points. The rigid body transformation from Velodyne coordinates to camera coordinates are given in detail at Zhirong et al. [32] and are expressed by:

$$T_{velo}^{cam} = \begin{pmatrix} R_{velo}^{cam} & t_{velo}^{cam} \\ 0 & 1 \end{pmatrix}$$

where $R_{velo}^{cam} \in R^{3\times3}$ is the rotation matrix and $t_{velo}^{cam} \in R^{1\times3}$ the translation vector from Velodyne to the camera coordinate system. The projection of a 3D point $x = (x, y, z, 1)^T$ in the lidar coordinates system gets projected to a point in the camera plane $y = (u, v, 1)^T$ according to:

$$y = T_{velo}^{cam} x$$

3.1.3.2 Perception Modules Correlation

Our purpose is the situational awareness improvement and the mitigation of cyber-attacks against computer vision systems, especially on the camera sensor. Firstly, we apply 2D semantic segmentation to the input image to recognize the objects in the scene. In a safe situation, the majority of the objects will be detected, whereas in an attacked case, some of them will be disappeared from the perception engine. In a parallel module, the 3D object detection model will be implemented, which has been trained to identify only the moving objects, which are mainly vehicles, pedestrians and cyclists. As soon as the coordinates of the 3D bounding boxes have been obtained, they are being projected to the image plane. The correlation is applied in the next step in order to fuse the 2D segmentation and 3D object detection outputs, as Figure 26 illustrates. In the latter, with a red mask and green bounding boxes, the 2D segmentation and 3D detection results are shown accordingly.

[32] Wu, Zhirong, et al. "3d shapenets: A deep representation for volumetric shapes." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

Figure 26: From top to bottom: 2D segmentation output and 3D detected objects projected to the image plane.

To correlate the two outputs, we isolate the region of the projected 3D bounding box to the image. We consider that the isolated region belongs to a specific class (vehicle, pedestrian, cyclist). We isolate respectively the same region from the segmentation mask. Finally, we compare the two outputs in order to estimate the overlap between the two detected regions. If an object has been detected by both of the modalities, the overlap should be high enough. Structural similarity index measure (SSIM) was used for the comparison and the threshold for a safe situation should be above 0.6 for the majority of the detected objects.

3.2 Multimodal SLAM

Simultaneous localization and mapping (SLAM) is a fundamental task for vehicle navigation in harsh and challenging environments (e.g. urban canyons, tunnels, underground parking, etc.), where the GPS sensor is highly unreliable. In general, SLAM aims to provide the pose of the vehicle as well as a detailed description of the environment in the form of a map. To do so, it relies on either the camera or LIDAR sensor, in order to develop and make use of Visual (VO) or LIDAR Odometry (LO) solutions, respectively. The six degrees of freedom pose contains the (x,y,z) translation and rotation (roll, pitch and yaw angles) of (camera or LIDAR) sensor, while the map is composed of 3D landmarks.

3.2.1 Methodology and approaches

SLAM methods can be distinguished to two main categories:
Visual odometry: To compute the local position and motion of a camera, VO algorithms must estimate the transformation that the camera undergoes between the current frame and a reference frame. The reference frame can be defined by the previous frame in the input sequence, some keyframe in the recent past, or a collection of frames from the recent past. In each case, the task is to estimate the transformation that takes information in the camera's current frame into the frame of reference of the past frame(s). This task can be seen as an optimization problem where the cost is given by the residual between the information measured in the current frame and corresponding information derived and reprojected from the reference frames. The vast majority of VO algorithms use feature-based

approaches[33]. The image is decomposed to a sparse set of interesting points, where for each interest point location the local appearance of the image is described by a feature vector that is invariant to camera transformations. The feature vectors are associated between the input frame and reference to form a set of geometric constraints from which we can derive the camera motion and scene structure. In this case, the cost function is formulated by the difference between the measured reprojection location of these interest points between frames, referred to as the reprojection error. However, some known limitations of feature-based methods include i) extraction of interesting points and feature vectors may be expensive (using well-known algorithms like SHIFT or SURF), ii) they are prone to errors in areas where there is a low number of interesting points, etc. On the contrary, dense or direct VO approaches[34,35] focus on minimizing the (geometric) reprojection error, aiming to directly minimize the photometric error between pixels in the optimization problem. State-of-the-art VO algorithms include Direct Sparse Odometry (DSO)[36], ORB-SLAM[37] and ORB-SLAM[38].

LIDAR odometry: LIDAR sensor provides dense 3D point clouds of vehicle's surroundings. The goal of LO is to estimate the pose of the vehicle by accumulating the transformation between consecutive frames of 3D point clouds. The existing LO solutions can be divided into two groups: point-wise and feature-wise methods. Point-wise methods estimate the relative transformation directly using the raw 3D points while feature-wise methods try to utilize more sophisticated characteristics of the point cloud such as the edge and planar feature points. The most well-known pointwise LO method is the iterative closest point (ICP)[39]. ICP operates at a point-wise level and directly matches two frames of the point cloud by finding the correspondences. One of the major drawbacks of the ICP is that when the frames include large quantities of points, ICP may suffer from a high computational load arising from the point cloud registration. Many variants of ICP have been proposed to improve its efficiency and accuracy, such as the Trimmed ICP[40] and Normal ICP[41]. To avoid the high computational load resulting from using the entire set of raw points, the feature-based LO methods extract a set of representative features from the raw points. The fast point feature histogram (FPFH) was proposed[42] to extract and describe important features. The FPFH enables the exploration of the local geometry and the transformation is optimized by matching the one-by-one FPFH-based correspondence. Another well-known feature-based LO method is the Low-drift and real-time lidar Odometry And Mapping LOAM[43]. Theoretically, LOAM integrates the properties of both the point-wise and the feature-wise methods. On the one hand, to decrease the computational load of typical ICP, LOAM proposed to extract two kinds of feature points, the edge and planar, respectively. The extraction of the feature is simply based on the smoothness of a small region near a given feature point. Different from the FPFH which provides multiple categories of features based on its descriptors, LOAM

[33] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007, pp. 225-234.
[34] F. Steinbrücker, J. Sturm and D. Cremers, "Real-time visual odometry from dense RGB-D images," 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011, pp. 719-722.
[35] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard and J. McDonald, "Robust real-time visual odometry for dense RGB-D mapping," 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 5724-5731.
[36] J. Engel, V. Koltun and D. Cremers, "Direct Sparse Odometry," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 3, pp. 611-625, 1 March 2018.
[37] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," in IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163, Oct. 2015.
[38] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017.
[39] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256.
[40] D. Chetverikov, D. Svirko, D. Stepanov and P. Krsek, "The Trimmed Iterative Closest Point algorithm," 2002 International Conference on Pattern Recognition, 2002, pp. 545-548 vol.3.
[41] J. Serafin and G. Grisetti, "NICP: Dense normal based point cloud registration," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 742-749.
[42] R. B. Rusu, N. Blodow and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 3212-3217.
[43] Zhang, Ji and Sanjiv Singh. “LOAM: Lidar Odometry and Mapping in Real-time.” Robotics: Science and Systems (2014).

involves only two feature groups. Another popular variant of LOAM is Lego-LOAM[44].

For the purposes of this deliverable, we have integrated in the CARLA autonomous driving simulator and deployed three state-of-the-art multi-modal odometry solutions, in order to evaluate their performance in realistic driving conditions: DSO, ORB-SLAM and Lego-LOAM. These approaches are discussed below:

DSO: DSO is a VO method based on a novel, highly accurate sparse and direct structure and motion formulation. It exploits a probabilistic model (minimizing a photometric error) with consistent, joint optimization of all model parameters, including geometry-represented as inverse depth in a reference frame-and camera motion. This probabilistic model takes noisy measurements Y as input and computes an estimator X for the unknown, hidden model parameters (3D world model and camera motion), following a Maximum A Posteriori (MAP) approach. Due to the direct formulation of DSO, it directly uses the actual sensor values-light received from a certain direction over a certain time period-as measurements Y in the probabilistic model. Additionally, one of the main benefits of a direct formulation is that it does not require a point to be recognizable by itself, thereby allowing for a more finely grained geometry representation (pixelwise inverse depth). Furthermore, data from across the image can be sampled—including edges and weak intensity variations generating a more complete model and lending more robustness in sparsely textured environments. A sparse framework (these methods use and reconstruct only a selected set of independent points, traditionally corners) has been chosen during the optimization since the main drawback of adding a geometry prior, as dense methods do, is the introduction of correlations between geometry parameters, which render a statistically consistent, joint optimization in real time infeasible. Optimization is performed in a sliding window, exploiting Gauss-Newton algorithm, where old camera poses as well as points that leave the field of view of the camera are marginalized. In contrast to existing approaches, this method further takes full advantage of photometric camera calibration, including lens attenuation, gamma correction, and known exposure times. This integrated photometric calibration further increases accuracy and robustness. DSO, apart from a geometric camera model which comprises the function that projects a 3D point onto the 2D image, considers also a photometric camera model, which comprises the function that maps real-world energy received by a pixel on the sensor (irradiance) to the respective intensity value. Furthermore, two important modules of DSO are Frame and Point Management. In terms of Frame Management, a window of up to N_f active keyframes (we use $N_f = 7$) must be kept. Every new frame is initially tracked with respect to these reference frames (Step 1). It is then either discarded or used to create a new keyframe (Step 2). Once a new keyframe-and respective new point are created, the total photometric error is optimized. Afterwards, one or more frames are marginalized (Step 3). As far as the Point Management is concerned, a fixed number N_p of active points (we use $N_p = 2000$), equally distributed across space and active frames, is used in the optimization. As a first step, N_p candidate points are identified in each new keyframe (Step 1). Candidate points are not immediately added into the optimization, but instead are tracked individually in subsequent frames, generating a coarse depth value which will serve as initialization (Step 2). When new points need to be added to the optimization, a number of candidate points (from across all frames in the optimization window) is chosen to be activated, i.e., added into the optimization (Step 3). Note that DSO only keeps N_p active points across all active frames combined.

ORB-SLAM: ORB-SLAM, is a feature-based monocular SLAM system that operates in real time, in small and large indoor and outdoor environments. The system is robust to severe motion clutter, allows wide baseline loop closing and relocalization, and includes fully automatic initialization. Building on excellent algorithms of recent years, a novel system that uses the same features for all SLAM tasks is designed: tracking, mapping, relocalization, and loop closing. ORB-SLAM system overview is shown in Figure 27 . ORB-SLAM extends the mapping capability to large scale

[44] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 4758-4765.

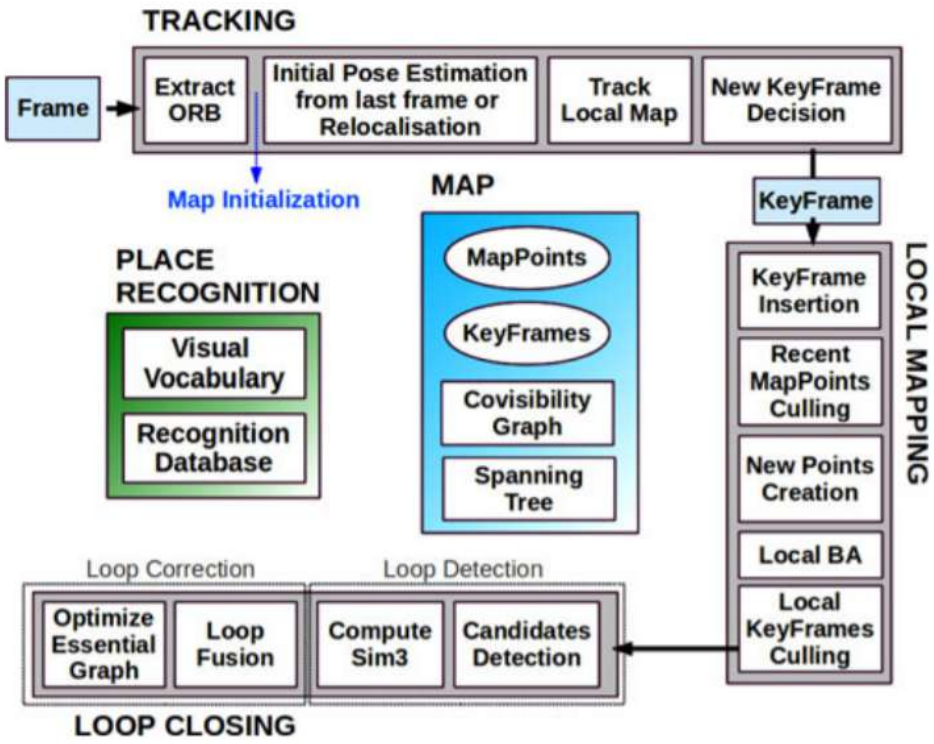


Figure 27: ORB-SLAM architecture[37]

spaces by dividing the mapping thread into a local mapping thread and a global mapping thread. The local mapping thread maintains a set of keyframes and landmarks which the tracking thread is used for camera pose estimation. However, in ORB-SLAM the local map is bounded in its complexity by only including keyframes that visually overlap with the current keyframe. During the tracking process correspondences between the input frame and the local map are computed. As keyframes are added to the local map, starting with the correspondences detected by the tracking thread, a further correspondence search is carried out to compute the level of visual overlap (i.e. shared features) between other keyframes in the local map. This information is stored in a co-visibility graph, where edges are added between camera nodes that share features. The weight of an edge corresponds to the number of shared features between its associated cameras. During operation the local mapping thread only considers keyframes linked to the current reference frame via edges above a threshold weight. As the camera explores, new keyframes are added liberally to the local map to ensure robust tracking, however in order to ensure complexity of the local mapping is bounded, keyframe and landmark culling strategies are also employed. Taken together, the co-visibility graph and the culling strategies decouple the complexity of the local mapping process from the scale of the global map. ORB-SLAM's large scale mapping capability is provided by a third thread. As each new keyframe is added to the co-visibility graph, the strongest associated edge is also added to a second essential graph. The strongest edge connects the new keyframe to the keyframe with the highest number of shared features. The essential graph therefore contains all of the keyframes from the co-visibility graph, but only a subset of the edges that form a spanning tree from the first keyframe, and any edges that are added due to loop closures. Therefore, as new frames are input from the camera they are first processed by the tracking thread which execute the following stages: i) ORB feature extraction, ii) Pose estimation and relocalization, iii) Local map tracking, and iv) Keyframe generation. As new keyframes are inserted the local mapping thread culls map and keyframes, adds new map points, and performs local bundle adjustment on the local map. In more detail, it executes the following steps: i) Keyframe insertion, ii) Map-point culling, iii) New point creation, iv) Local Bundle Adjustment, v) Keyframe culling. Finally, as new keyframes are generated the loop closing thread attempts to detect loop closures as follows: i) Place recognition & loop detection, ii) 7DOF (translation, rotation and scale) constraint estimation, iii) Loop fusion and iv) Pose graph optimization.

Lego-LOAM: Lightweight and Ground-Optimized LOAM (LeGO-LOAM) is a LO solution for pose estimation in complex environments with variable terrain. LeGO-LOAM is lightweight, as real-time pose estimation and mapping can be achieved on an embedded system. Point cloud segmentation is performed to discard points that may represent unreliable features after ground separation. LeGO-LOAM is also ground-optimized, as a two-step optimization for pose estimation is introduced. Planar features extracted from the ground are used to obtain z translation, roll and pitch during the first step. In the second step, the rest of the transformation (x,y translation and yaw) is obtained by matching edge features extracted from the segmented point cloud. The ability to perform loop closures to correct motion estimation drift is also introduced. Lego-LOAM system overview is shown in Figure 28. The overall system is divided into five modules. The first, segmentation, takes a single scan's point cloud and projects it onto a range image for segmentation. The segmented point cloud is then sent to the feature extraction module, which determines two types of features: edge and planar. Then, LIDAR Odometry uses features extracted from the previous module to find the transformation relating consecutive scans using the two-step Levenberg-Marquardt optimization. The features are further processed in LIDAR mapping, which registers them to a global point cloud map. At last, the transform integration module fuses the pose estimation results from lidar odometry and LIDAR mapping and outputs the final pose estimate. The proposed system seeks improved efficiency and accuracy for ground vehicles, with respect to the original, generalized LOAM framework.

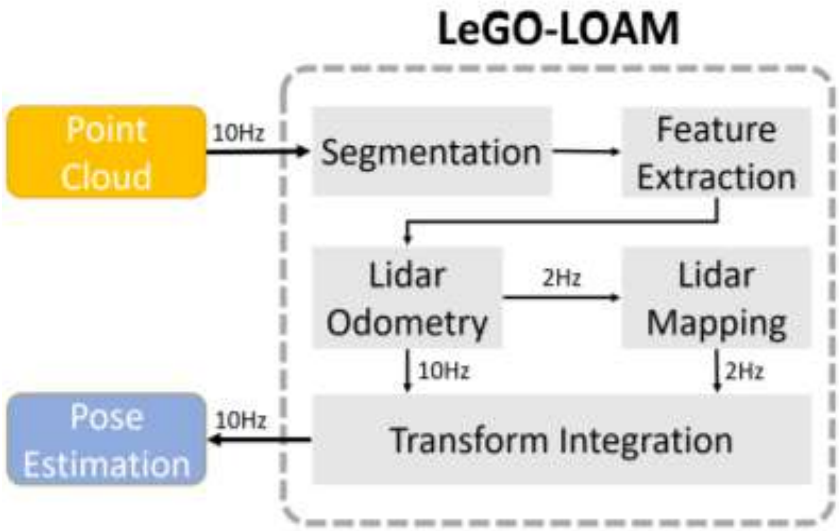


Figure 28: Lego-LOAM architecture[44]

3.2.2 Simulation orchestration and setup

3.2.2.1 Simulation Orchestration

The Human in the loop control in a Single Vehicle scenario describes a situation where a vehicle is being monitored by a number of sensors which record the drivers state and behavior while performing several referential tasks, some of which are stated below:

- lane keeping performance
- mean speed
- pedal activity
- steering behavior
- mean lateral position

The CPSoSAware platform encapsulates a number of simulation tools which can be exploited to emulate not only Autonomous Vehicle and Advanced Driver Assistance Systems (AV/ADAS) scenarios, as well as instances of an individual

handler in command of a vehicle in a road instance scenario. Each simulator is equipped with sensors that produce data that can be then exploited by the CPSoSAware framework. CARLA is for example one of the open-source simulators that can be exploited to that end, and its setup is explained in paragraph 3.2.2.2 below. Hence, by collecting data in a controlled environment we can create a baseline to use as a reference in a practical use case in a real-world environment.

The process of setting up these tools consists of several scripts that need to be configured to service the parameters set by each possible scenario. To be able to control the setup of this procedure in a centralized and easily scalable manner, the Simulation Block component of the CPSoSAware platform exploits the Orchestrator developed in T2.5 and described in Deliverable 2.2. This platform performs the automation of the aforementioned procedures, and implements the setup of the simulation scenarios from a single endpoint in a Graphical User Interface (GUI) that collects all simulators and all their configurations. The workflow of this process is shown in Figure 29 below.

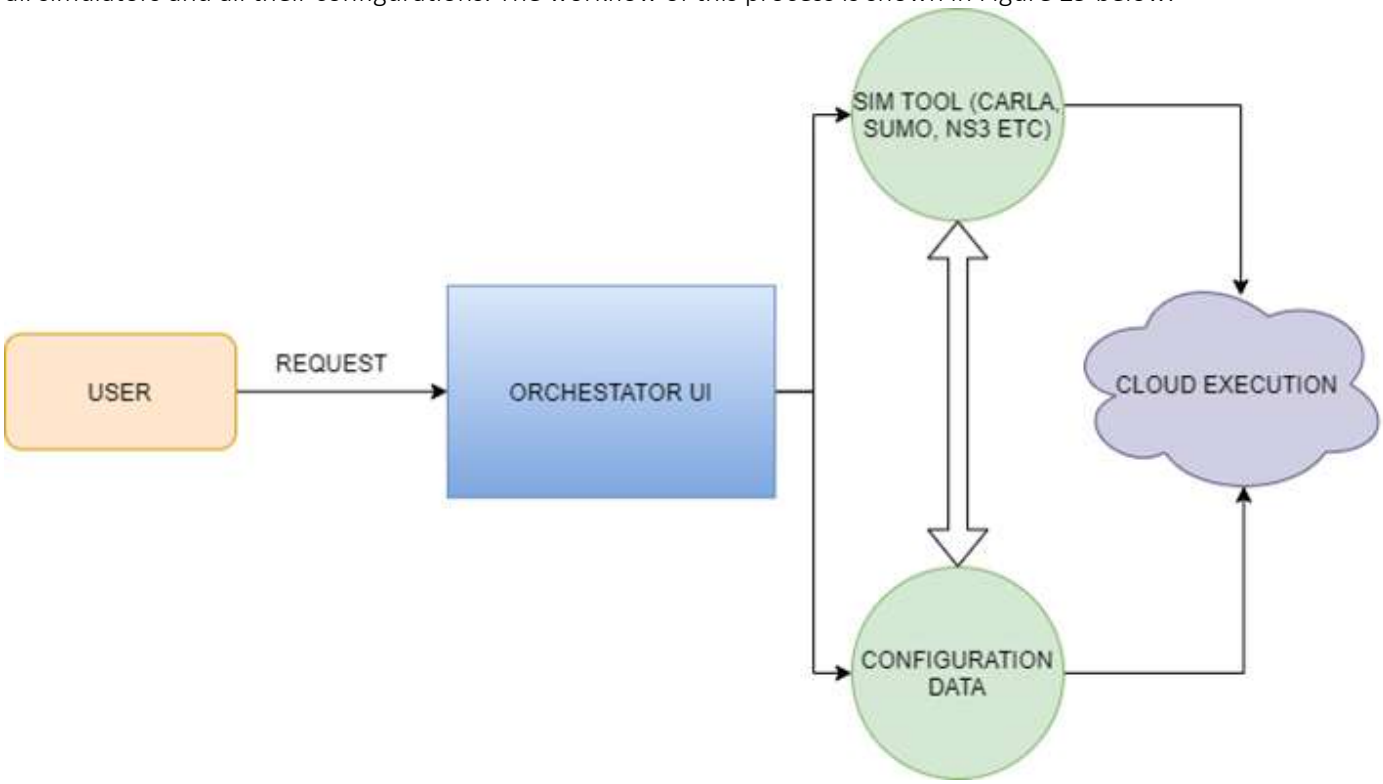


Figure 29: Orchestrator workflow.

3.2.2.2 Simulation setup

The setup of the simulation framework is illustrated in the following image. It is mainly composed of a data producer, namely the Carla simulator, and various data consumers in the form of ROS nodes. The connection between the virtual environment and the ROS runtime is established via the carla-ros bridge which amongst others is responsible for translating the sensor data derived from Carla to ROS compliant messages and vice versa. More specifically, the components of the simulation infrastructure are listed below.

Carla

CARLA is an open-source autonomous driving simulator. It is based on Unreal Engine for conducting the simulation and utilizes the OpenDrive standards for defining targets and urban settings. Simulation parameters can be controlled programmatically via a C++ or Python API. Moreover, it consists of scalable client-server architecture, in which the server is responsible for everything having to do with the simulation, like scene rendering, updating physics, actors'

state etc.

ROS Bridge

The ROS-Bridge facilitates the bidirectional communication between the simulation environment and ROS runtime. The messages from CARLA are translated to relevant ROS topics and at the same time messages originating from ROS get translated to CARLA commands.

ROS Nodes

ROS Nodes are executables that use ROS to communicate with other nodes by implementing a publish/subscribe scheme. The aforementioned Odometry algorithms were implemented as ROS Nodes. In addition, already available auxiliary nodes, like RVIZ, were used for visualization or new ones were developed, like Logger, which is responsible for writing simulation data in csv files.

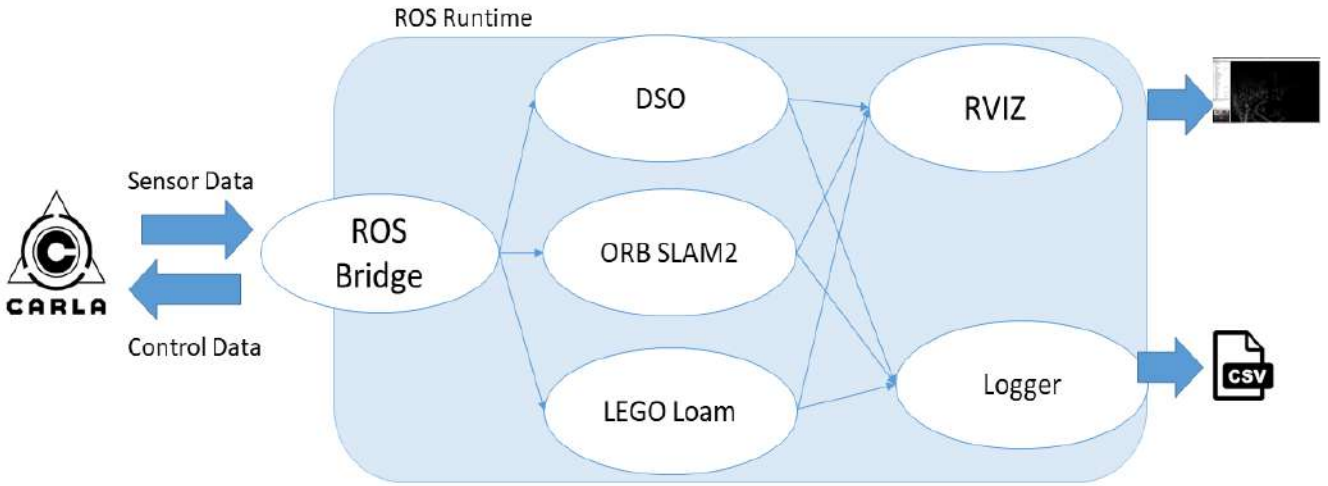


Figure 30: Simulation Framework Setup.

For running the test scenarios, we had selected the map called Town10HD which is a feature rich urban environment. We have tested the accuracy of the proposed algorithms under different weather and lighting conditions. For measuring the accuracy of the algorithms, we have selected two widely accepted and used metrics, the Absolute Trajectory Error and Relative Path Error.

The ego vehicle is a common vehicle which is equipped with a rgb camera and a lidar sensor. The rgb camera provides 800x600 images at 30Hz. The lidar sensor simulates a Velodyne-16 lidar, which supports 16 channels and produces approximately 300,000 points/sec rotating at 10Hz.

For evaluating[45] the estimated trajectories we have used evo which is a python library which provides executables and

[45] https://github.com/MichaelGrupp/evo

a library for handling, comparing and estimating the produced trajectory of odometry and SLAM algorithms.

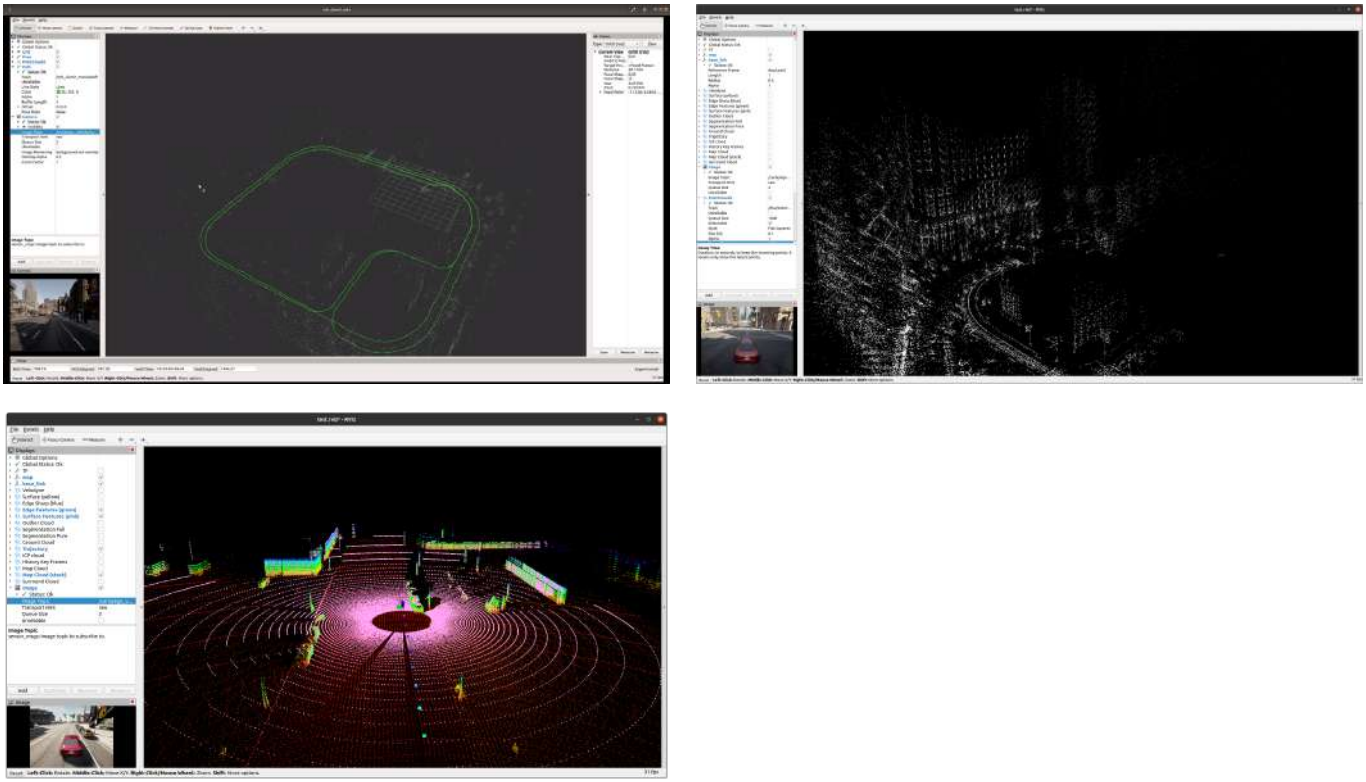


Figure 31: Rviz visualizing the outputs of ORB SLAM2 (upper left corner), DSO (upper right corner), and Lego LOAM (top left corner).

3.2.3 Results

Evaluation metrics for the different multi-modal odometer solutions include[45] Root Mean Square Error (RMSE), Mean, Median, Standard Deviation, Minimum and Maximum Absolute Trajectory Error (ATE), both translational and rotational. Figure 32 , Figure 33 and Figure 34 represent corresponding screenshots from CARLA simulator, alongside with driving and simulation parameters.
The following tables contain indicative results of the discussed multi-modal SLAM approaches, under 3 different scenarios:

Table 5: Scenario No1 (target ego velocity: 40km/h, weather: clear, light: day).

Algorithm	RMSE	ATE MEAN	ATE MEDIAN	ATE STD	ATE MIN	ATE MAX
DSO	1.735	1.476	1.215	0.912	0.132	3.165
ORB-SLAM2	1.598	1.435	1.162	0.704	0.249	3.399
LEGO-LOAM	2.19	1.979	1.006	0.940	0.04	4.629

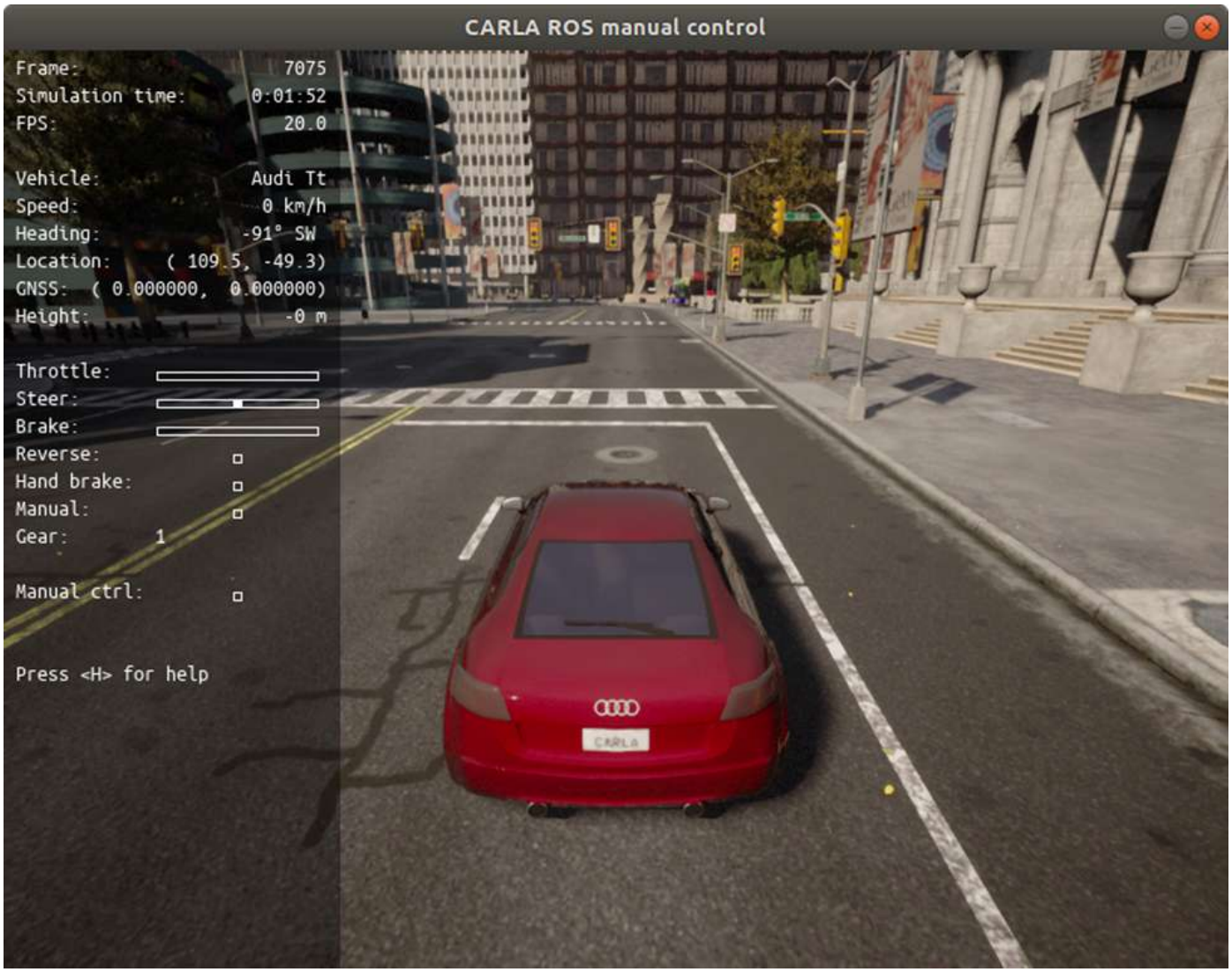


Figure 32: CARLA screenshot for Scenario No1.

Table 6: Scenario No2 (target ego velocity: 40km/h, weather: rainy, light: day).

Algorithm	RMSE	ATE MEAN	ATE MEDIAN	ATE STD	ATE MIN	ATE MAX
DSO	3.561	3.268	2.978	1.416	0.522	6.7
ORB-SLAM2	4.121	3.995	3.118	2.114	0.325830	8.225
LEGO-LOAM	2.591	2.10	1.643	1.507	0.162	6.449

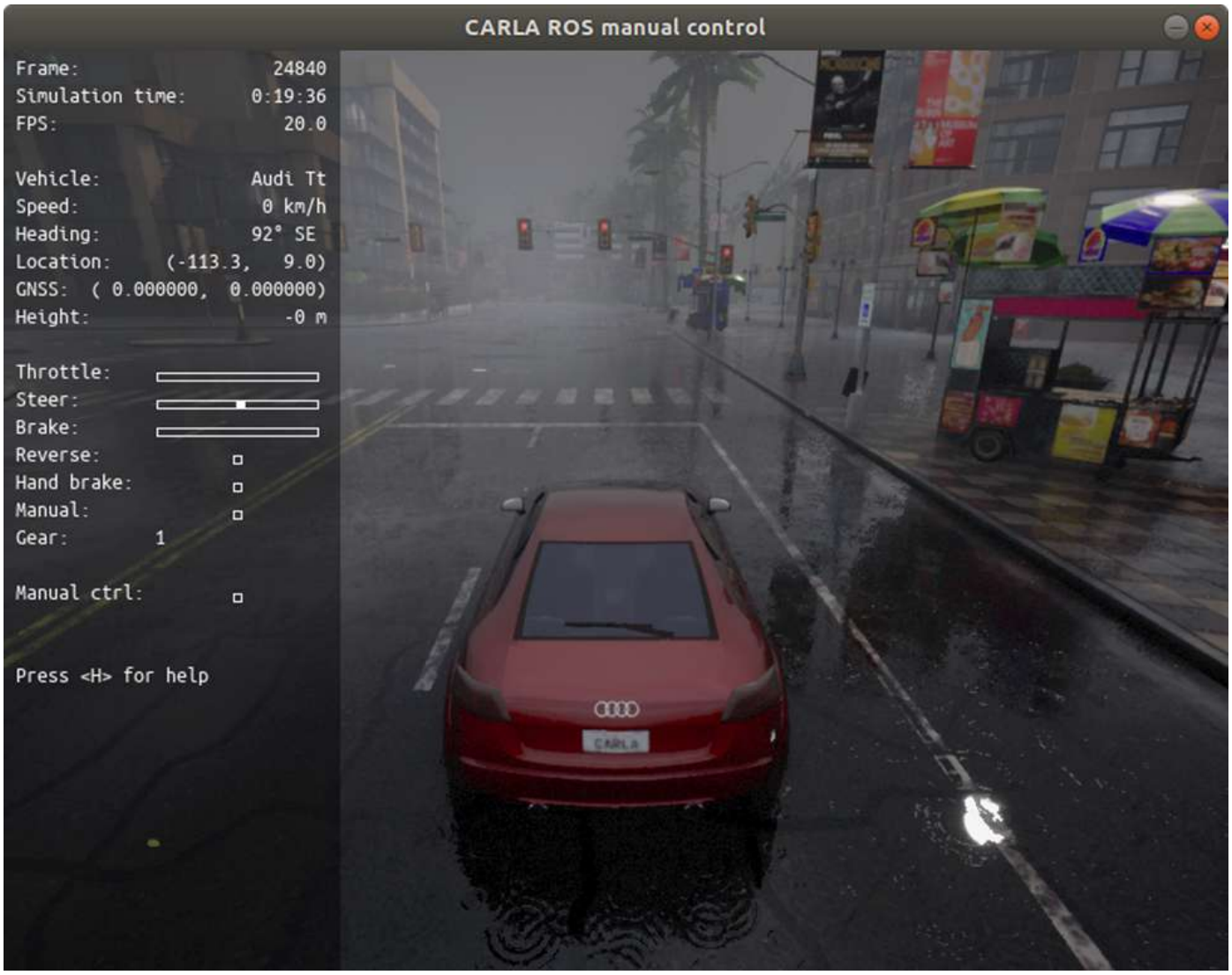


Figure 33: CARLA screenshot for Scenario No2.

Table 7: Scenario No3 (target ego velocity: 40km/h, weather: clear, light: night).

Algorithm	RMSE	ATE MEAN	ATE MEDIAN	ATE STD	ATE MIN	ATE MAX
DSO	3.123	3.109	3.56	0.273	2.826	4.448
ORB-SLAM2	2.921802	2.326664	1.811241	1.767360	0.227186	6.718740
LEGO-LOAM	4.408	3.639	3.273	1.774	0.574	11.807

Figure 34: CARLA screenshot for Scenario No3.

4 Compression and acceleration techniques for scene analysis Deep Neural Networks

In this Section, we focus on more elaborate and high-performing MCA techniques that belong to weight sharing[46,47,48] and study their impact on the performance of object detection for autonomous driving, based on on both 2D and 3D detectors.

4.1 Problem Formulation

The core operation performed by a convolutional layer and the involved quantities, are depicted Figure 35. In particular,

[46] S. Nousias, E. Pikoulis, C. Mavrokefalidis, A. S. Lalos, and K. Moustakas, “Accelerating 3D scene analysis for autonomous driving systems,” in IEEE International Conference on Very Large-Scale Integration, 2021.
[47] S. Nousias, E. V. Pikoulis, C. Mavrokefalidis, and A. S. Lalos, “Accelerating deep neural networks for efficient scene understanding in automotive cyber-physical systems,” IEEE Int. Conf. Ind. Cyber-Physical Syst., 2021.
[48] Cheng, Jian, et al. "Quantized CNN: A unified approach to accelerate and compress convolutional networks." IEEE transactions on neural networks and learning systems 29.10 (2017): 4730-4743.

the input volume consists of N channels $\mathbf{X}_i$, $i = 1,2,\dots,N$. Also, there are M kernel volumes and the k-th kernel volume has N filters $\mathbf{W}_{k,i}$, $k = 1,2,\dots,M$, $i = 1,2,\dots,N$. For simplicity, it is assumed that the dimensions of the $\mathbf{X}_i$'s, $\mathbf{W}_{k,i}$'s and $\mathbf{U}_k$'s are $m \times m$, $p \times p$, and $m \times m$, respectively.
The convolution of the input volume with the k-th kernel volume is given by

$$\mathbf{U}_k = \sum_{i=1}^{N} \mathbf{X}_i \star \mathbf{W}_{k,i}, \qquad (1)$$

where $\star$ denotes the 2D convolution operation.
In order to proceed and describe the entities to be clustered (i.e., coded by the codebook that will be designed), (1) is re-written in order to describe the (i,j)-th element $\mathbf{U}_k[i,j]$ as

$$\mathbf{U}_k[i,j] = \sum_{n,l \in \mathcal{R}_{i,j}} \mathbf{x}_{n,l}^T \mathbf{w}_{k,i-n,j-l}, \qquad (2)$$

where $\mathbf{x}_{i,j} = \left[\mathbf{X}_1[i,j],\dots,\mathbf{X}_N[i,j]\right]^T$ contains the samples at the (i,j)-th position of all input channels. Also, $\mathbf{w}_{k,u,v} = \left[\mathbf{W}_{k,1}[u,v],\dots,\mathbf{W}_{k,N}[u,v]\right]^{\mathrm{T}}$ contains the filter weights at the (u,v) position of all channels in the k-th kernel volume. Finally, the set $\mathcal{R}_{i,j}$ contains p^2 indices around the position (i,j).
In the product quantization framework, the N-dimensional vector space is partitioned into S, N'-dimensional subspaces with $N' = N/S$, so that the s-th subspace spans dimensions $[(s-1)N'+1,\dots,sN']$, $s = 1,\dots,S$. Let us now partition vectors $\mathbf{x}_{i,j}$, $\mathbf{w}_{k,u,v}$ defined in (2), accordingly, as

$$\begin{aligned} \mathbf{x}_{i,j} &= \left[\left(\mathbf{x}_{i,j}^1\right)^{\mathrm{T}},\dots,\left(\mathbf{x}_{i,j}^S\right)^{\mathrm{T}}\right]^{\mathrm{T}}, \\ \mathbf{w}_{k,u,v} &= \left[\left(\mathbf{w}_{k,u,v}^1\right)^{\mathrm{T}},\dots,\left(\mathbf{w}_{k,u,v}^S\right)^{\mathrm{T}}\right]^{\mathrm{T}}, \end{aligned} \qquad (3)$$

where each of the sub-vectors lies in N'-D space. Then, (2) can be rewritten as

$$\mathbf{U}_k[i,j] = \sum_{s=1}^{S} \sum_{n,l \in \mathcal{R}_{i,j}} \left(\mathbf{x}_{n,l}^s\right)^T \mathbf{w}_{k,i-n,j-l}^s, \qquad (4)$$

where the inner sum denotes the contribution of the s-th subspace to the k-th convolutional output, at position (i,j). For each subspace, the goal of product quantization is to perform vector quantization to the Mp^2 kernel sub-vectors lying in s-th subspace, and cluster them into $K_s \ll Mp^2$ clusters. This way, each sub-vector is represented by the centroid of the cluster it belongs to, reducing accordingly the number of required dot-products. To be more specific, the acceleration occurs because the original dot-products between the input and the Mp^2 kernel sub-vectors, are approximated by the ones between the input and the K_s centroids/representatives.

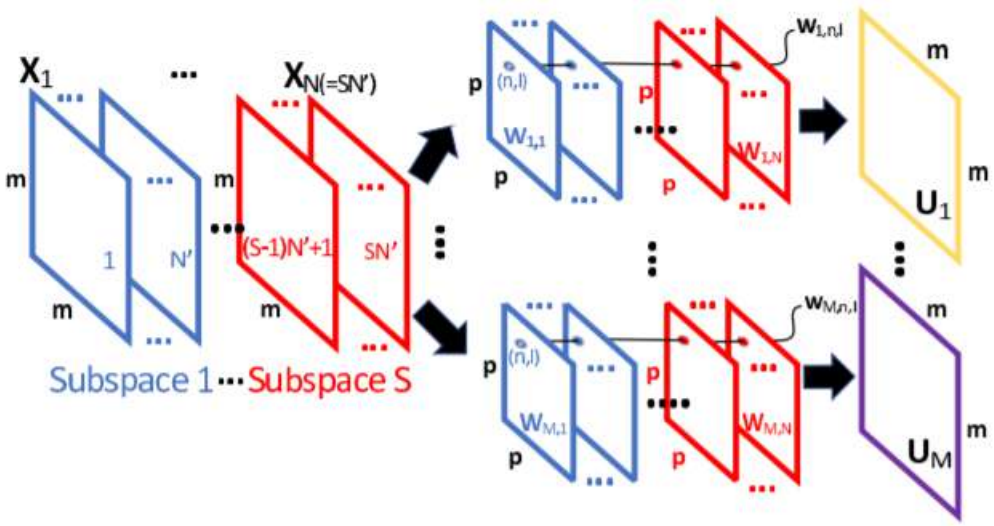


Figure 35: Linear operation of a single convolutional layer[49]

4.2 Dictionary-learning-based weight clustering

In this section, first, the proposed codebook structure for approximating the kernel sub-vectors, is described and discussed in comparison with the conventional codebook structure that appears in current literature. This discussion is also extended towards the gains that are achieved through a computational complexity analysis. Then, the proposed codebook design is approached as a Dictionary Learning (DL) problem, which actually treats the k-means-based conventional codebook design as a special case. The latter will be referred to as Vector Quantization (VQ) in the following. Finally, some implementation details are described concerning the initialization of the involved parameters when applying the proposed DL solution.

4.3 Proposed approximation

Let us first define the kernel approximation scheme incurred by the conventional codebook structure, as follows:

$$\mathbf{W} \approx \mathbf{C\Gamma}, \tag{5}$$

where the columns of $\mathbf{W} \in \mathbb{R}^{N\prime \times p^2 M}$, $\mathbf{C} \in \mathbb{R}^{N\prime \times K_{vq}}$, and $\mathbf{\Gamma} \in \mathbb{R}^{K_{vq} \times p^2 M}$, contain the kernel sub-vectors (of a particular subspace), the representatives (or cluster centroids), and assignment vectors, respectively. Specifically, each column of $\mathbf{\Gamma}$ has exactly one non-zero element, equal to 1, meaning that each column of $\mathbf{W}$ is approximated by one column of $\mathbf{C}$. Thus, in the conventional case, the Mp^2 sub-vectors are approximated by $K_{vq} \ll p^2 M$ representatives, using the codebook $\mathbf{C}$.

Instead, in this paper, the following approximation is proposed:

$$\mathbf{W} \approx \mathbf{D\Lambda\Gamma}, \tag{6}$$

where $\mathbf{W} \in \mathbb{R}^{N\prime \times p^2 M}$ and $\mathbf{\Gamma} \in \mathbb{R}^{K_{dl} \times p^2 M}$ are defined as in (5), while $\mathbf{D} \in \mathbb{R}^{N\prime \times L_{dl}}$ and $\mathbf{\Lambda} \in \mathbb{R}^{L_{dl} \times K_{dl}}$ denote the dictionary and the matrix of sparse coefficients, respectively. Specifically, the columns of $\mathbf{D}$ (called dictionary atoms), are normalized, while $\mathbf{\Lambda}$ is a sparse matrix in the sense that each of its columns contains at most α non-zero elements, with α being the sparsity level. Thus, under the proposed scheme, the $p^2 M$ sub-vectors are approximated via K_{dl} representatives contained in the codebook $\mathbf{D\Lambda}$. In turn, these representatives are obtained as linear combinations of

[49] Pikoulis E.V., Mavrokefalidis C., Nousias S., Lalos A.S. (2022) A New Clustering-Based Technique for the Acceleration of Deep Convolutional Networks. In: Wani M.A., Raj B., Luo F., Dou D. (eds) Deep Learning Applications, Volume 3. Advances in Intelligent Systems and Computing, vol 1395. Springer.

at most α atoms from a dictionary of size L_{dl}, with $L_{dl} < K_{dl} \ll p^2 M$. Note that the matrix approximation defined in (6) can be viewed as a special case of the general Dictionary Learning (DL) problem[50] , which is why we call our acceleration technique as a DL-based one.

It should be noted that, in the general case, the proposed approximation requires more representatives than the conventional approach (i.e., $K_{dl} > K_{vq}$), for achieving the same quantization (i.e. approximation) error. This is by definition since the conventional codebook $\mathbf{C}$ is obtained in an unconstrained fashion, while the proposed codebook $\mathbf{D\Lambda}$ follows a specific structure. Although it seems counter intuitive (in the sense that the proposed approximation is less efficient than the conventional one, in the general case), due to the particularities of the problem at hand, namely, due to the fact that the "data points" in $\mathbf{W}$ are in fact filters used in convolution operations, the proposed approximation results actually in significantly higher acceleration ratios for the same quantization error, as it is going to be demonstrated. This is because, due to the linearity of the operations performed in the convolutional layer, the sparse coefficients in $\mathbf{\Lambda}$ need only be applied to the convolution between the input and the dictionary atoms in $\mathbf{D}$, instead of the atoms themselves. This endows the proposed approximation scheme with the flexibility to use a number of representatives K_{dl} that is several times larger than K_{vq}, while restricting the size of the dictionary (so that $L_{dl} \ll K_{vq}$) thus reducing the number of "heavy" convolutions, as it will become clearer in the following subsection.

4.4 Computational complexity analysis

Since the core operations of a DNN are ultimately translated into dot-products between input and kernel vectors, the computational complexity of a DNN is usually measured in terms of the number of Multiply and Accumulate (MAC) operations. A MAC is dominated by the involved multiplication (MUL), which is a significantly "heavier" computation than the involved addition. As such, in the subsequent analysis, in order to compare the techniques on a common ground, MAC and MUL operations are going to be used interchangeably, i.e, a MAC will be considered equivalent to one MUL, so that the computational cost is measured as the number of MULs.

Let us first begin by examining the computational complexity of the original layer, where, by arranging the m^2 input sub-vectors of the s-th subspace in the columns of a matrix $\mathbf{X} \in \mathbb{R}^{N' \times m^2}$, we see that the convolution operation involves the calculation of a matrix product of the form:

$$\mathbf{Y} = \mathbf{X}^T \mathbf{W}, \tag{7}$$

where $\mathbf{W}$ contains the kernel sub-vectors (as defined in (5)), followed by the appropriate summation of the dot-products according to (4). Since calculating $\mathbf{Y}$ requires $m^2 p^2 M N'$ multiplications, the overall (i.e. for all S subspaces) computational complexity of the original convolutional layer, measured in MULs, is obtained as:

$$\mathcal{T}_o = m^2 p^2 M (S N') = m^2 p^2 M N. \tag{8}$$

In the VQ case (described by the approximation in

(9)), the approximate $\mathbf{Y}$ is obtained as:

$$\mathbf{Y} \approx (\mathbf{X}^T \mathbf{C}) \mathbf{\Gamma}, \tag{9}$$

namely, it involves calculating the dot-products between input sub-vectors and representatives and then "plugging" the results appropriately, according to the columns of $\mathbf{\Gamma}$. Calculating $\mathbf{X}^T \mathbf{C}$ requires only $m^2 N' K_{vq}$ MULs (as $k_{vq} \ll M p^2$), meaning that the overall computational complexity for the approximate convolutional output is reduced to:

$$\mathcal{T}_{vq}(K_{vq}) = m^2 (S N') K_{vq} = m^2 N K_{vq}.$$

[50] B. Dumitrescu and P. Irofti, Dictionary Learning Algoithms and Applications. Springer, 2018.

(10)

Finally, for the DL-based approximation scheme, we can write:

$$\mathbf{Y} \approx ((\mathbf{X}^T\mathbf{D})\mathbf{\Lambda})\mathbf{\Gamma},$$

(11)

meaning that, in this case, calculating the approximate $\mathbf{Y}$ is a two-stage operation. First, we calculate $\mathbf{X}^T\mathbf{D}$, i.e., the dot-products between the input and the dictionary atoms, which requires $m^2N'L_{dl}$ MULs, where L_{dl} denotes the dictionary size. Subsequently, the results are combined according to the columns of $\mathbf{\Lambda}$, which requires $\alpha\, m^2K_{dl}$ additional MULs Thus, the overall computational complexity for the approximate convolutional output in the DL case, is obtained as:

$$\mathcal{T}_{dl}(K_{dl}, L_{dl}, \alpha) = m^2(NL_{dl} + \alpha SK_{dl}).$$

(12)

Accordingly, the acceleration ratio (namely, the ratio of original vs accelerated computational complexities) achieved by the two rival approaches, can be written as follows:

$$\begin{aligned} \rho_{vq} &\equiv \frac{\mathcal{T}_o}{\mathcal{T}_{vq}} = \frac{p^2M}{K_{vq}} \\ \rho_{dl} &\equiv \frac{\mathcal{T}_o}{\mathcal{T}_{dl}} = \frac{p^2M}{L_{dl}+\frac{\alpha}{N'}K_{dl}} \end{aligned}$$

(13)

Of great interest is also the relative acceleration between the proposed DL-based and the VQ approach, which will also provide rules for selecting the free parameters of the proposed technique. First, in order to have a better representation error, we set the number of representatives used by the proposed technique as a multiple of the representatives used by the VQ approach, i.e., $K_{dl} = c\,K_{vq}$, $c > 1$. Then, using (10), (12), we can see that for the DL-based approximation to achieve at least the same acceleration with the VQ technique, i.e., for $\mathcal{T}_{dl}(K_{dl}, L_{dl}, \alpha) \leq \mathcal{T}_{vq}(K_{vq})$ to hold, the following inequality should hold regarding the size of the used dictionary:

$$L_{dl} \leq K_{vq}\left(1 - \frac{\alpha\, c}{N'}\right).$$

(14)

As we are going to demonstrate in our experimental results for various combinations of the coefficient c and sparsity level α, and for (14) holding with equality (i.e., for the two rivals achieving the same acceleration ratio), the proposed technique leads to a significantly better approximation of the original weights, which ultimately translates into better classification accuracy for the accelerated DNNs.

4.5 Proposed algorithm

For deriving the matrix factorizations described by the proposed weight factorization in (6), the quantization error between the original $\mathbf{W}$ and its approximate version is minimized. In particular, the following minimization problem is defined.

$$\begin{aligned} \min_{\mathrm{D},\Lambda,\Gamma} \quad & \| \mathbf{W} - \mathbf{D}\mathbf{\Lambda}\mathbf{\Gamma} \|_F^2 \\ \text{s.t.} \quad & \|\mathbf{d}_i\|_2^2 = 1, i = 1, \dots, L_{dl}, \\ & \|\lambda_i\|_0 \leq \alpha, i = 1, \dots, K_{dl}, \\ & \|\gamma_i\|_0 = 1, 1^{\mathrm{T}}\gamma_i = 1, \quad i = 1 \dots Mp^2, \end{aligned}$$

(15)

where $||\cdot||_F$, $||\cdot||_2$, $||\cdot||_0$ denote the Frobenius, l_2, and l_0 norms, respectively, while the last constraint ensures that the elements of $\mathbf{\Gamma}$ take values in $\{0,1\}$ and each of its columns has exactly one non-zero element.
In order to solve (15), we follow a strategy of alternating optimizations over each set of parameters, leading to the

following three sub-problems:

Sparse coding

With $\mathbf{D}$, $\mathbf{\Gamma}$ fixed, the loss function in (15) can be rewritten as follows:

$$||\mathbf{W} - \sum_{i=1}^{K_{dl}} (\mathbf{D}\boldsymbol{\lambda}_i)\tilde{\boldsymbol{\gamma}}_i||_F = \sum_{i=1}^{K_{dl}} ||\mathbf{W}_{I_i} - (\mathbf{D}\boldsymbol{\lambda}_i)\mathbf{1}^T||_F, \quad (16)$$

where $\mathbf{y}_i$, $\tilde{\mathbf{y}}_i$ is used to denote the i-th column and row of matrix $\mathbf{Y}$, respectively, $I_i = \{j|\gamma_{ij} = 1\}$ is the set of indices of the non-zero elements of $\tilde{\boldsymbol{\gamma}}_i$, $\mathbf{W}_{I_i}$ is the submatrix formed by the columns of $\mathbf{W}$ indexed by I_i, while $\mathbf{1}$ denotes the all-ones vector of dimension $|I_i|$.
Observing (16), due to the l_0-norm constraints on $\mathbf{\Gamma}$, I_i's, $i = 1, \dots, K_{dl}$, are a partition of $\{1,2, \dots, p^2M\}$, meaning that the minimization of (16) over $\mathbf{\Lambda}$ is translated into K_{dl} separate sub-problems, one for each of $\mathbf{\Lambda}$'s columns:

$$\begin{aligned} \min_{\zeta} \quad & \left\|\mathbf{W}_{I_t} - (\mathbf{D}\zeta)1^{\mathrm{T}}\right\|_F^2 \\ \text{s.t.} \quad & \| \zeta \|_0 \leq \alpha. \end{aligned} \quad (17)$$

In order to solve (17), we follow an Orthogonal Matching Pursuit (OMP) approach, which builds the support of the sparse representation (non-zero elements of $\boldsymbol{\lambda}_i$), by adding one dictionary atom at a time, up to α atoms[50].
This sparse coding sub-problem is outlined in lines 5-12 of Table 8. There, $\mathcal{S}$ denotes a set of non-zero indices, while $\mathbf{D}_{\mathcal{S}}$ and $\boldsymbol{\zeta}_{\mathcal{S}}$ contain the columns of $\mathbf{D}$ and the elements of $\boldsymbol{\zeta}$ indexed by $\mathcal{S}$, respectively.

Dictionary update

With $\mathbf{\Lambda}$, $\mathbf{\Gamma}$ fixed, we write the loss function in (15) as follows:

$$\mathcal{E} = ||\mathbf{W} - \mathbf{DG}||_F^2 = ||\mathbf{W} - \sum_{i=1}^{L_{dl}} \mathbf{d}_i\tilde{\mathbf{g}}_i||_F^2, \quad (18)$$

where $\tilde{\mathbf{g}}_i$ denotes the i-th row of $\mathbf{G} = \mathbf{\Lambda\Gamma}$. Thus, the dictionary update step translates to minimizing $\mathcal{E}$ under the l_2-norm constraint for the dictionary atoms. In order to solve this problem, we follow the coordinate-descent-based approach outlined in Algorithm 3.5[50]. This sub-problem is described in lines 13-18 of Table 8.

Assignment update

With $\mathbf{D}$, $\mathbf{\Lambda}$ fixed, the loss function in (15) takes the following form:

$$\mathcal{E} = ||\mathbf{W} - \tilde{\mathbf{C}}\mathbf{\Gamma}||_F^2 = \sum_{i=1}^{p^2M} ||\mathbf{w}_i - \tilde{\mathbf{C}}\boldsymbol{\gamma}_i||_2^2 \quad (19)$$

where $\tilde{\mathbf{C}} = \mathbf{D\Lambda}$ is the $N' \times K_{dl}$ matrix of representatives. Taking into account the special structure of $\mathbf{\Gamma}$, updating $\boldsymbol{\gamma}_i$ is equivalent to determining the position j_i of its non-zero (unity) element, which simply assigns $\mathbf{w}_i$ to its closest representative. This sub-problem is described in lines 19-21 of Table 8.

Table 8: Proposed algorithm for solving (15).

1: **procedure** DL-based sub-space clustering
2: **Input**: original sub-vectors $\mathbf{W}$, # of representatives K_{dl} dictionary size L_{dl}, sparsity level α.
3: Obtain initial solution $\{\mathbf{D}_0, \mathbf{\Lambda}_0, \mathbf{\Gamma}_0\}$
4: **repeat**
5: **for** $i = 1: K_{dl}$ //*Sparse Coding*
6: **Initialize**: $\mathbf{E} = \mathbf{W}_{I_i}$, $\mathcal{S} = \emptyset$
7: **for** $j = 1: \alpha$
8: Build new support: $\mathcal{S} \leftarrow \mathcal{S} \cup \{k\}$,

where $k = \text{argmax}_{j \notin \mathcal{S}} |\mathbf{1}^T \mathbf{E}^T \mathbf{d}_j|$.
9: Find new solution: $\boldsymbol{\zeta}_{\mathcal{S}}$ by solving $\min_{\boldsymbol{\xi}} ||\mathbf{W}_{I_i} - \mathbf{D}_{\mathcal{S}}\, \boldsymbol{\xi}\, \mathbf{1}^T||_F^2$.
10: Update residual: $\mathbf{E} = \mathbf{W}_{I_i} - \mathbf{D}_{\mathcal{S}}\, \boldsymbol{\zeta}_{\mathcal{S}}\, \mathbf{1}^{\mathrm{T}}$.
11: **end**
12: **end**
13: **Initialize**: $\mathbf{E} = \mathbf{W} - \mathbf{D}\boldsymbol{\Lambda}\boldsymbol{\Gamma}$ //*Dictionary Update*
14: **for** $i = 1: L_{dl}$
15: Modify error: $\mathbf{F} = \mathbf{E}_{I_i} + \mathbf{d}_i\, \tilde{\mathbf{g}}_{i,I_i}$.
16: Update i-th atom: $\mathbf{d}_i = \frac{\mathbf{F}\,(\tilde{\mathbf{g}}_{i,I_i})^T}{||\mathbf{F}\,(\tilde{\mathbf{g}}_{i,I_i})^T||}$.
17: Re-compute error: $\mathbf{E}_{I_i} = \mathbf{F} - \mathbf{d}_i\, \tilde{\mathbf{g}}_{i,I_i}$.
18: **end**
19: **for** $i = 1: p^2 M$ //*Assignment Update*
20: Update $\boldsymbol{\gamma}_i$ solving $j_i = \text{argmin}_{j \in \{1,\dots,K_{dl}\}} ||\mathbf{w}_i - \tilde{\mathbf{c}}_j||_2$.
21: **end**
22: **Until**: a maximum number of iterations is met.
23: **Return**: $\mathbf{D}$, $\boldsymbol{\Lambda}$, $\boldsymbol{\Gamma}$.
24: **end procedure**

4.6 Initial Solution and Parameter Selection

In order to provide an initial solution $\mathbf{D}_0$, $\boldsymbol{\Lambda}_0$, $\boldsymbol{\Gamma}_0$, to the proposed acceleration technique, we work as follows:

1. We obtain a clustering of the original kernel sub-vectors into K_{dl} clusters by minimizing

$$\min_{\mathbf{C},\boldsymbol{\Gamma}} \quad ||\mathbf{W} - \mathbf{C}\boldsymbol{\Gamma}||_F \qquad (20)$$

under the constraints on the assignment matrix $\boldsymbol{\Gamma}$ stated in (15). This problem can be solved by using the k-means algorithm.

2. We then obtain a sparse representation of the cluster centroids in

$$\mathbf{C} \approx \mathbf{D}_0 \boldsymbol{\Lambda}_0, \qquad (21)$$

by using a dictionary of size L_{dl} and target sparsity α. This problem can be solved with standard DL techniques such as the ones described earlier.

3. Finally, we obtain the initial assignment matrix $\boldsymbol{\Gamma}_0$ by assigning each of the sub-vectors in $\mathbf{W}$ to its closest representative in $\tilde{\mathbf{C}} = \mathbf{D}_0 \boldsymbol{\Lambda}_0$.

There are four free parameters in the proposed technique, namely, the subspace dimension N', the number of representatives K_{dl}, the size of the dictionary L_{dl}, and the sparsity level α.

For a target acceleration ρ, we first determine the number of representatives K_{vq} required by the VQ technique in order to achieve ρ by using (14). This provides a lower bound for the number of representatives K_{dl} required by the proposed technique. We set $K_{dl} = c\, K_{vq}$, $c > 1$. Then, for a target sparsity α, we use (16) with equality in order to determine the dictionary size required to achieve ρ. Typical ranges for the parameter values are $c = 2, \dots, 5$, $\alpha = 1,2,3$, and $N' = 4, \dots, 8$.

4.7 Application on image-based object detection

Both networks were trained with the KITTI odometry dataset. Three classes are taken into account, namely, cyclists, pedestrians and cars which were manually annotated with bounding boxes containing the objects in the scene. A significant observation regarding the dataset is that not all objects of the same class are labeled in each and every image. Such a fact plays a role in the evaluation of the detection outcome as our analysis will reveal. The dataset was split in a 80%, 20% for training and validation, respectively, resulting in $N_{tr} = 5980$ training examples and $N_{val} = 1497$ validation examples.

As we have mentioned before in Section 3.1.1.2, for the training of the SqueezeDet architecture, Stochastic Gradient Descent (SGD) was employed with the following values for the hyperparameters (determined via experimentation); batch size B=8, learning rate $LR = 10^{-4}$, with a weight decay rate $D_W = 10^{-4}$, a learning rate decay rate of $D_{LR} = 2 * LR/N_e$, number of steps $N_s = 3 \times N_{tr}$ and a dropout rate of 50%, over a total of $N_e = 300$ epochs. Training and testing took place in an NVIDIA GeForce GTX 1080 graphics card with 8GB VRAM and compute capability 6.1 in a Intel(R) Core(TM) i7-4790 CPU @ 3.60Hz based system with 32GB of RAM.

Likewise, for Resnet50ConvDet, we also employed SGD with hyperparameter values as in the case of SqueezeDet. Training and evaluation of Resnet50ConvDet took place in an NVIDIA GeForce Geforce RTX 2080 with 16GB VRAM and compute capability 7.5 in a Intel(R) Core(TM) i7-4790 CPU @ 3.60Hz based system with 16GB of RAM.

In all cases, training took place with a data augmentation scheme where the bounding boxes drift by $k_x * 150$ pixels across the x-axis and $k_y * 150$ pixels across the y-axis, where $k_x, k_y \sim U(0,1)$. A 50% probability is also assumed to flip the object.

4.7.1 Acceleration scheme

In our experiment, we apply the rival techniques to the two detection models in a "full-model" acceleration scenario. It involves accelerating multiple (or all) convolutional layers of the original models and measuring the achieved performance of the accelerated networks.

It is noted, here, that, although full-range acceleration depends heavily on the performance of the technique used for the acceleration of each layer, it also involves experimentation over the strategy used for accelerating the layers and the involved fine-tuning (re-training) of the accelerated model. Here, we follow a stage-wise acceleration approach (as proposed in Cheng et al.[51]) with each stage involving the acceleration (and fixing) of one or more layers of the network, and, subsequently, fine-tuning (i.e., re-training) the remaining original layers. The starting point for each stage is the accelerated and fine-tuned version of the previous stage. The process begins with the original network and it is repeated until all target layers are accelerated. For fine-tuning and performance assessment, we use the training and validation datasets from KITTI, as previously explained.

4.7.2 Accelerating SqueezeDet

The feature-extraction part of SqueezeDet, namely SqueezeNet, is responsible for roughly 83% of the total 5.3×10^9 multiply-accumulate (MAC) operations required. Since SqueezeNet constitutes an already "streamlined" network, in our acceleration experiments we followed a moderate acceleration strategy only targeting the "expand". Acceleration was performed in a one-module-per-stage fashion for a total of 8 acceleration stages. Using acceleration ratios of α=8, 10, 12, and 20 on the targeted layers, an acceleration of the SqueezeNet part by 72%, 74%, 75%, and 78%, respectively, and a total model acceleration by 59%, 60%, 62%, and 65%, respectively, were achieved.

4.7.3 Accelerating ResNetDet

The feature extraction part of ResNetDet is responsible for roughly 81% of the total 3.5×10^10 MACs required by the

[51] Cheng, Jian, et al. "Quantized CNN: A unified approach to accelerate and compress convolutional networks." IEEE transactions on neural networks and learning systems 29.10 (2017): 4730-4743.

network. Following the network's architecture, in our experiments with ResNetDet, we accelerated its convolutional (feature-extraction) blocks in a one-block-per-stage fashion leading to 13 total acceleration stages. Using acceleration ratios of α=8, 10, 12, and 20 on the targeted layers, an acceleration of the feature-extraction part by 84%, 86%, 88%, and 92%, respectively, and a total model acceleration by 67%, 69%, 71%, and 74%, respectively, were achieved.

4.7.4 Metrics

For each detection, the Intersection Over Union (IOU) score is computed as the ratio of area of intersection to the area of union between the predicted and ground-truth bounding boxes. A true positive occurs when IOU>0.5 and the predicted class is the same as the ground-truth class. A false positive occurs when IOU<0.5 or a different class is detected, meaning that unmatched bounding boxes are taken as false positives for a given class. Precision, recall and mean average precision (mAP) are subsequently calculated according to Powers et al. [52].

4.7.5 Results

The progressive, stage-wise acceleration results for the employed networks, using both the VQ and the DL acceleration techniques for various acceleration ratios, are shown in Figure 36. The rightmost point in every plot depicts the performance of the "fully" accelerated network, i.e., after all targeted convolutional layers have been accelerated. At each point, the performance of the detectors was assessed based on the achieved mean average precision (mAP) and recall.

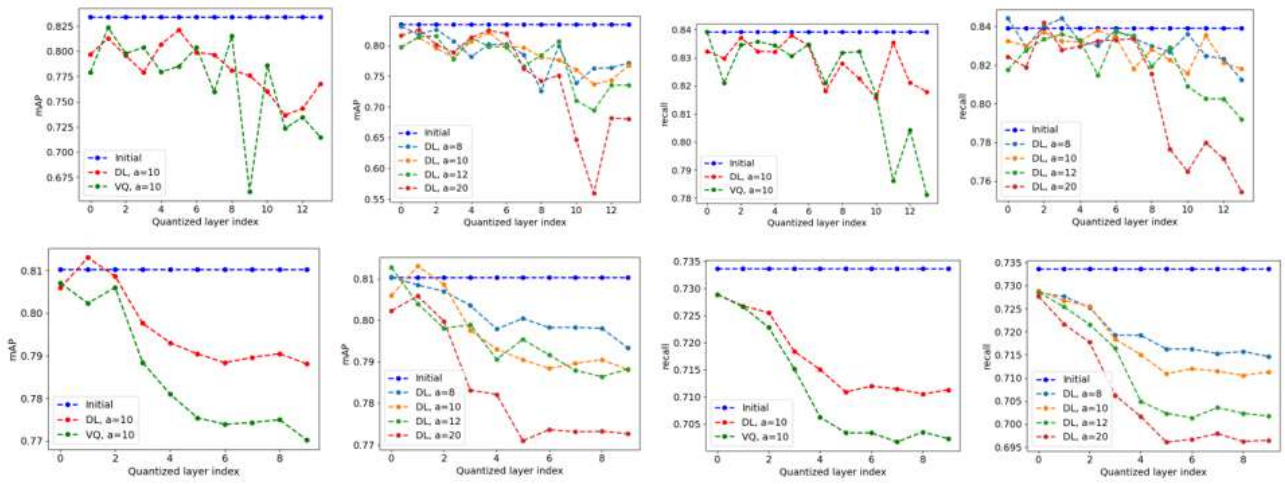


Figure 36: Performance evaluation and comparison of DL vs VQ acceleration techniques on ResNetDet (top row) and SqueezeDet (bottom row).

Table 9: Average precision (AP) and Recall (RC) scores for the original and accelerated ResNetDet, SqueezeDet models.

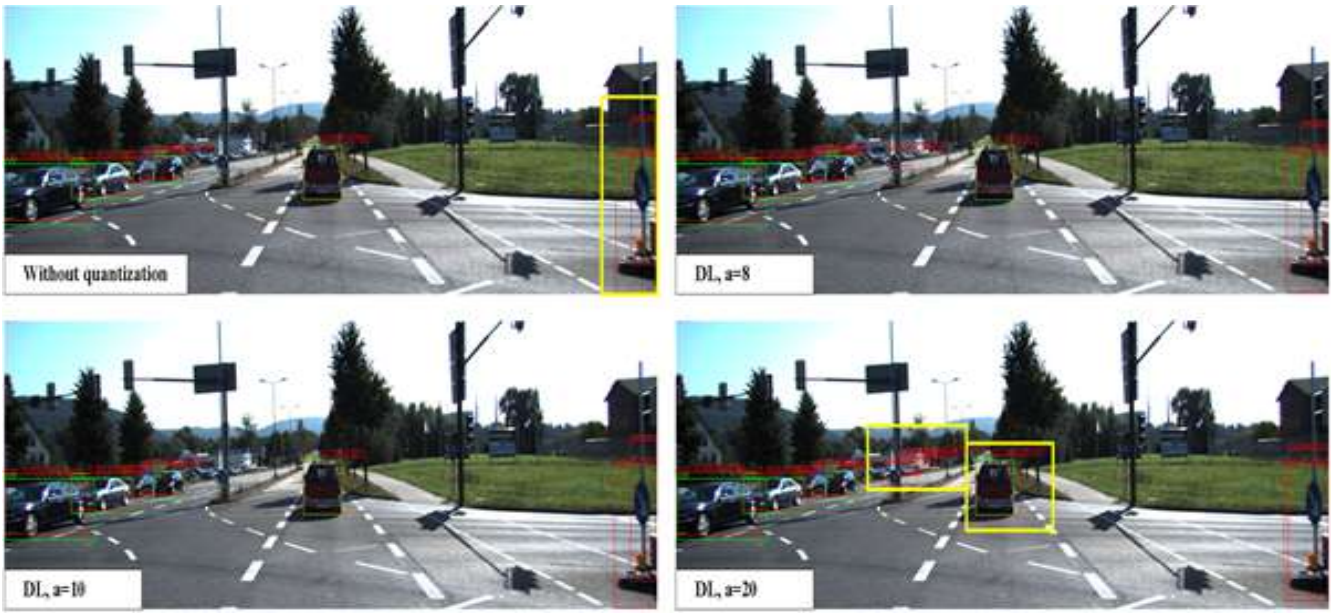
SqueezeDet						
Class	Car		Cyclist		Pedestrian	
Score (%)	*AP*	*RC*	*AP*	*RC*	*AP*	*RC*
Original	88.9	73.3	78.5	54	75.6	56.6
DL ($\alpha = 8$)	88.1	71.1	76.5	51.5	73.3	52.8
DL ($\alpha = 10$)	87.9	71.1	74.7	48.4	73.7	51.8
DL ($\alpha = 12$)	87.4	70.1	76.3	46.2	72.7	51.5
DL ($\alpha = 20$)	87.1	69.6	73.1	46.2	71.4	48.3

[52] Powers, David MW. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." arXiv preprint arXiv:2010.16061 (2020).

VQ ($\alpha = 10$)	87.5	70.2	74	46.8	69.4	49.2

ResNetDet						
Class	Car		Cyclist		Pedestrian	
Score (%)	*AP*	*RC*	*AP*	*RC*	*AP*	*RC*
Original	94.3	83.9	78.6	66.2	77.7	66.8
DL ($\alpha = 8$)	90.4	82.6	71.8	64.3	70.7	62.1
DL ($\alpha = 10$)	90.1	81.9	71.2	61.9	69.2	60.4
DL ($\alpha = 12$)	88.5	79.2	68.4	58.5	64.7	59.8
DL ($\alpha = 20$)	85.6	75.7	61.2	46.9	59.7	55.1
VQ ($\alpha = 10$)	88.2	78.4	64.3	54.1	64.1	57.4

As a general comment, the results presented in Figure 36 reveal a very promising performance by the employed weight-sharing techniques, and, especially so, for the DL-based one, whose application results in significantly accelerated detectors, with limited loss of their detection capabilities, as expressed by both the mAP and recall values. It should be also noted that these results could be further improved by following a more targeted acceleration strategy (e.g., via experimentation over the acceleration sequence, the acceleration ratio per layer, using a more extensive fine-tuning process, etc.) which acts as further confirmation of our conclusion. Moreover, comparatively speaking, the DL-based technique managed to generally outperform its rival in our experiments, as highlighted by the plots presented in Figure 36, for an acceleration ratio of a=10 (Figure 37 (a)&(c), and (e)&(g), for ResNetDet, and SqueezeDet, respectively).



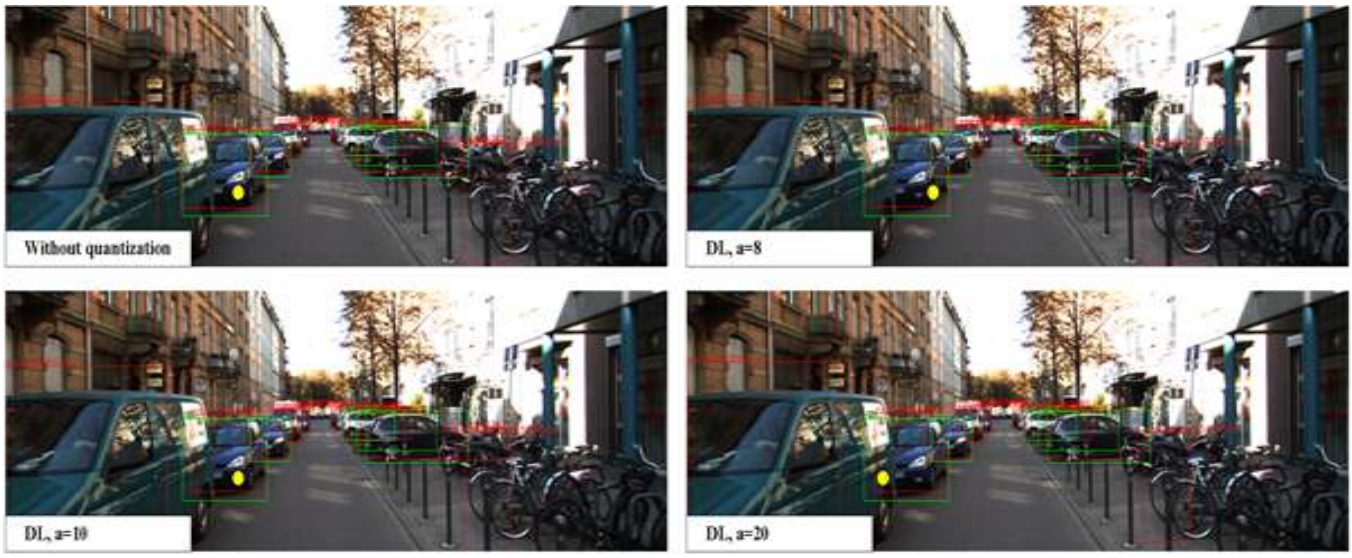


Figure 37: Application of accelerated vs original SqueezeDet models, using examples from the KITTI dataset. Green rectangles correspond to ground truth boxes, while red rectangles to predictions. The confidence scores are also shown in red letters. Yellow rectangles in (a) and yellow dot in (b) highlight the most obvious performance degradation of the accelerated networks, as compared to the original one.

4.7.6 Error-type analysis

For a better insight on the obtained results, we performed an in-depth analysis of the error-types of the employed detectors. For this analysis, we examined 64 images containing the ground truth annotation and the detection outcome and classified the errors into seven categories; a) object located but not labeled in dataset, b) object located buy bounding box not in place (IOU<0.5), c) object located but overlapping double bounding box appeared, d) nonexistent object located, e) object not located due to occlusion, f) object not located at all, and g) mirrored object (i.e., on glass surface), object located but in wrong class. Furthermore, we manually classified the 64 images into clear scenes with sufficient light and no occlusions, and messy scenes with many objects some of them being occluded. The motivation behind this perspective is that the detector Catalink detects an object, but it is assumed as an error or the detector Catalink misses an object (i.e., occlusion) but it is assumed as an error since it was originally annotated in the dataset. As we can observe 50% of the errors in the examined images, are objects that were actually found but either they were not annotated or there was a bounding box issue. The results of this qualitative analysis are summarized by the bar-charts shown in Figure 38.

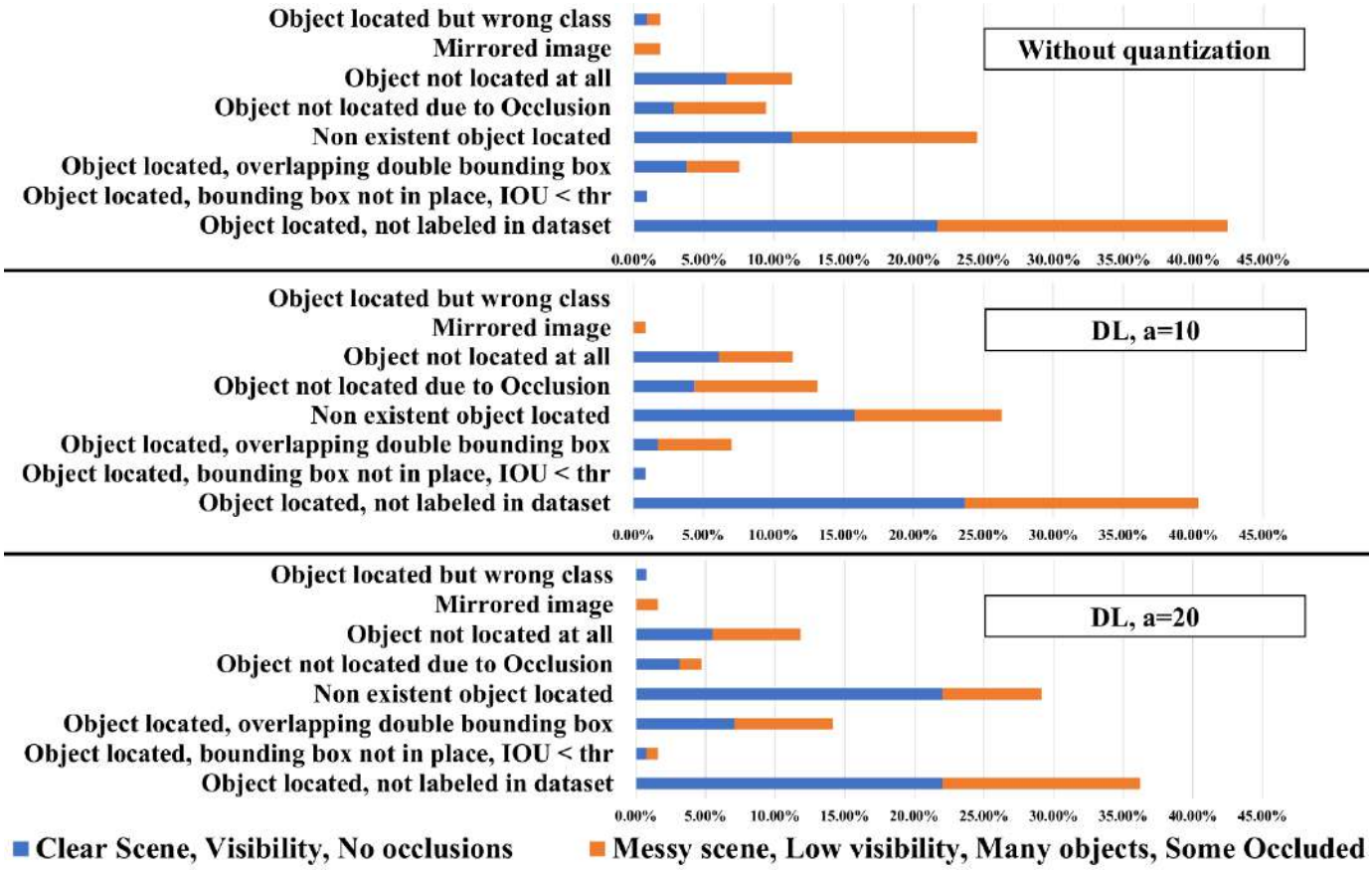
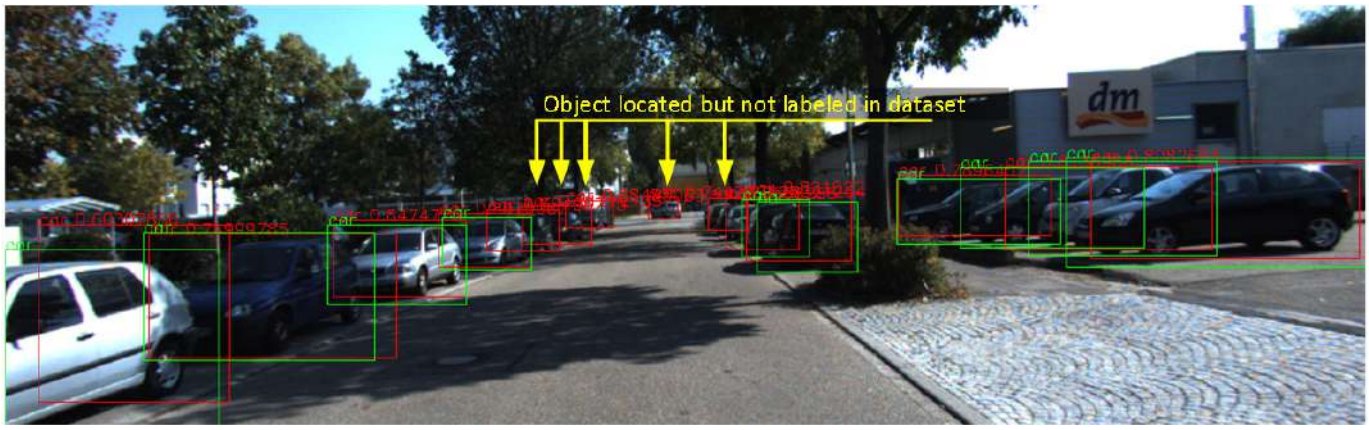


Figure 38: Error classification for original models, DL with a=10 case and DL case with a=20.



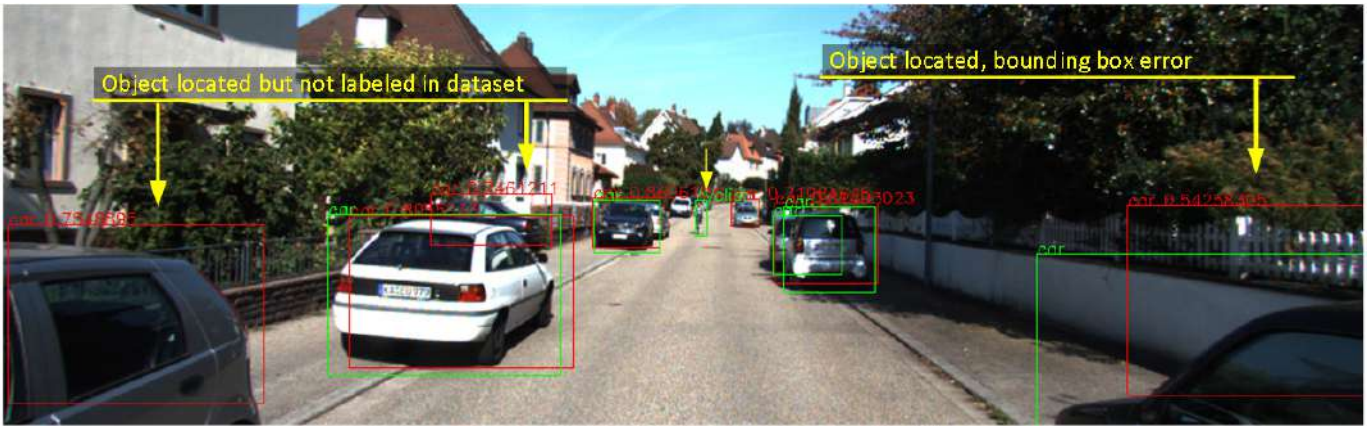


Figure 39: Examples of error types. Green boxes refer to ground truth data, while red boxes to detections.

4.8 Application on LIDAR based object detection

Both networks were trained with the KITTI 3D object detection benchmark consisting of 7481 training images and 7518 test images as well as the corresponding point clouds, comprising a total of 80.256 labelled objects In our study, three classes are mainly examined, cyclists, pedestrians and cars annotated with bounding boxes containing the objects in the 3D scene. As we mentioned before in Section 3.1.2, 3716 annotated Velodyne point cloud scenes were used for training and 3769 annotated Velodyne point cloud scenes were used for testing and validation. For the deployment and retraining of PointPillars and PV-RCNN, the OpenPCDet[53] framework was employed. For the initial evaluation, pre-trained instances were used, while for the retraining, the Adam optimizer was employed with learning rate $l_r = 0.003$, weight decay rate $D_W = 10^{-2}$ and a batch size $B = 4$. Training took place in an NVIDIA Geforce RTX 2080 with 16GB VRAM and compute capability 7.5. Furthermore, for the Pointpillars network the detection accuracy was evaluated on NVIDIA Jetson TX2, while for the PV-RCNN network, due to model size, the detection accuracy was evaluated on the NVIDIA Geforce RTX 2080.

In our experiments, we apply the VQ and DL weight-sharing techniques to the PontPillars and PV-RCNN models, targeting their convolutional layers, and measuring the performance drop induced by the acceleration, compared to the original networks. The reported acceleration ratios are defined as the ratio of the original to the accelerated computational complexities, measured by the number of multiply-accumulate (MAC) operations. To achieve our acceleration goal, we followed the stage-wise strategy, whereby the individual layers are accelerated progressively in stages, starting from the original network. At each stage, the parameters of one or more layers are quantized using the presented techniques and fixed, and subsequently, the remaining layers are re-trained to adapt to the newly presented changes. The process is then repeated for the convolutional layers involved in the next stage, and so on until all desired layers are accelerated. The KITTI 3D object detection dataset is employed for the fine-tuning and performance evaluation, ensuring that the same training examples that were used during the initial training are also used during the fine-tuning step.

4.8.1 Accelerating PointPillars

PointPillars is a fully convolutional network with its feature-extraction part (both 2D and transposed convolution operators) being responsible for 97.7% of the total MAC operations required. In total Pointpillars network encompasses 4.835×10^6 parameters and require 63.835×10^9 MACs. For a good balance between acceleration and performance drop, we targeted the 2D convolutional layers of PointPillars (consuming approximately 47% of the total MACs), as well as the 4×4 transposed convolutional layer of the network (responsible for 44.4% of the total MACs), depicted with the red blocks in Figure 25(a). Acceleration was performed in 16 acceleration stages with each stage involving the quantization of a particular layer, followed by fine-tuning. Using acceleration ratios of $\alpha = 10, 20, 30$, and

[53] OD Team. "Openpcdet: An open-source toolbox for 3d object detection from point clouds." (2020).

40 on the targeted layers, lead to a reduction of the total required MACs by 82%, 86%, 88%, and 89%, or equivalently, to total model acceleration of PointPillars by $5.6\times$, $7.6\times$, $8.6\times$, and $9.2\times$, respectively.

4.8.2 Accelerating PV-RCNN

The main bulk of the operations required by PV-RCNN are consumed by the Voxel-Backbone and the BEV-Backbone blocks shown in Figure 25(b), with the former one being composed of Submanifold Sparse 3D-Conv layers[54], while the latter consisting of regular 2D convolutional layers. Since the Sparse convolutional layers are already specialized layers that are designed to exploit the sparsity of the input to reduce their computational complexity and keeping in mind that the number of operations required by such layers is input-dependent, in this experiment we focused only on the BEV-Backbone block of PV-RCNN, as shown in Figure 25(b). PV-RCNN network encompasses 12.405×10^6 parameters and requires 88.878×10^9 MACs without taking into account the sparse convolutional layers.
In this case, the targeted layers (highlighted in Figure 25(b)) are responsible for roughly 86% of the MACs required by the BEV-Backbone block. Similarly, to the previous experiment, using acceleration ratios of α = 10, 20, 30, and 40 on the targeted layers, lead to a reduction of the MACs required by the BEV-Backbone block by 77%, 82%, 83%, and 84%, or equivalently, to the block's acceleration by $4.5\times$, $5.5\times$, $6.0\times$, and $6.3\times$, respectively.

4.8.3 Metrics

The official KITTI evaluation detection metrics include bird eye view (BEV), 3D, 2D, and average orientation similarity (AOS). The 2D detection is done in the image plane and average orientation similarity assesses the average orientation (measured in BEV) similarity for 2D detections. The KITTI dataset is categorised into easy, moderate, and hard difficulties, and the official KITTI leaderboard is ranked by performance on moderate. For the sake of self-completeness, easy difficulty refers to a fully visible object with a minimum bounding height box of 40px and max truncation of 15%, moderate difficulty refers to a partially occluded object with a minimum bounding box height of 25px and max truncation of 30% and hard difficulty refers to a difficult to see an object with a minimum bounding box height of 40px and max truncation of 50%. Each 3D ground truth detection box is assigned to one out of three difficulty classes (*easy, moderate, hard*), and the used 40-point Interpolated Average Precision metric is separately computed on each difficulty class. It formulated the shape of the Precision/Recall curve as $AP|_R = 1|R|\sum_{r\in R} \quad \rho_{interp}(r)$ averaging the precision values provided by $\rho_{interp}(r)$, according to Simonelli et al.[55] . In our setting, we employ forty equally spaced recall levels, $R_{40} = \{1/40,2/40,3/40,\dots,1\}$. The interpolation function is defined as $\rho_{interp}(r) = \underset{r':r'\geq r}{max}\rho(r')$, where $\rho(r)$ gives the precision at recall r, meaning that instead of averaging over the actually observed precision values per point r, the maximum precision at recall value greater or equal than r is taken.

4.8.4 Object detection

In this section, the impact of the VQ and DL acceleration techniques on the performance of PointPillars and PV-RCNN is presented, following the procedure outlined in Sec. 4. Table 10 summarizes the average precision (AP) for various acceleration ratios in the case of PointPillars. For each category (namely, car, cyclist, pedestrian), the three AP correspond to the three levels of difficulty (namely, easy, moderate and hard) that the evaluation dataset provides. It is observed, as expected, that the impact on performance is greater as the acceleration ratios are increased, a tendency also observed for other values of acceleration ratios that are not depicted here. The relative performance drop is at most 1%, 3% and 5% for the three difficulty levels (easiest to hardest) of "car", while it is at most 18%, 21% and 21% for "pedestrian" and 11%, 15% and 16% for "cyclist". It is observed that these results are promising when the two weight-sharing techniques are employed, as the aforementioned maximum performance drops correspond to a considerable reduction on MAC operations, while the impact is negligible for "car". Note that these acceleration gains

[54] Graham, Benjamin, and Laurens van der Maaten. "Submanifold sparse convolutional networks." arXiv preprint arXiv:1706.01307 (2017).
[55] Simonelli, Andrea, et al. "Disentangling monocular 3d object detection." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019

can be further enhanced by more tailored MCA configurations involving e.g. layer-specific compression/acceleration ratios. Finally, an example depicting false positive errors from the application of the original and the accelerated networks is shown in Figure 40. For PV-RCNN, the performance of the model remains practically unaffected by the acceleration of the targeted layers, as shown by the results summarized in Table 11. This can be attributed to the limited extent of the affected part of the network, but also to the quality of the employed acceleration techniques. Moreover, it is interesting that applying the procedure described in Sec. 4.2, which involves progressively isolating and fine-tuning specific portions of the network, has even proven beneficial for the overall network's performance, for certain combinations of categories/acceleration ratios.

Table 10: Point Pillars: BEV Average Precision. The shown acceleration ratios of α = 10, 20, 30, and 40 on the targeted layers, corresponds to a total acceleration of PointPillars by 5.6×, 7.6×, 8.6×, and 9.2×, respectively.

	Car			Pedestrian			Cyclist		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Original	92.04	88.06	86.66	61.60	56.01	52.05	85.26	66.24	62.22
DL, α = **10**	91.83	87.32	84.72	56.45	51.00	46.93	82.66	63.05	58.87
DL, α = **20**	92.03	87.40	84.73	56.46	50.18	46.22	83.03	64.53	60.62
DL, α = **30**	91.76	86.80	84.08	53.49	47.14	43.71	78.82	60.61	56.98
DL, α = **40**	92.57	85.10	83.67	51.04	44.79	41.91	78.81	58.82	55.02
VQ, α = **10**	93.08	87.24	84.54	56.74	51.02	47.28	82.78	63.83	59.89
VQ, α = **20**	91.90	86.74	84.20	54.11	47.84	43.82	84.34	64.50	60.31
VQ, α = **30**	91.11	84.77	81.88	54.64	47.64	43.67	76.42	59.50	55.66
VQ, α = **40**	91.75	86.41	82.44	50.81	44.94	41.53	75.26	56.31	52.83

Table 11: PV-RCNN: BEV Average Precision. The presented acceleration ratios of α = 10, 20, 30, and 40 on the targeted layers, corresponds to a total acceleration of the BEV-Backbone block by 4.5×, 5.5×, 6.0×, and 6.3×, respectively.

	Car			Pedestrian			Cyclist		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Original	94.51	90.19	88.09	71.12	64.12	59.89	91.35	74.63	69.89
DL, α = **10**	95.67	90.46	88.34	68.95	60.84	56.24	92.81	76.99	72.39
DL, α = **20**	94.83	90.21	88.17	65.03	58.20	54.67	92.81	74.18	69.67
DL, α = **30**	95.63	90.40	88.37	66.33	58.74	54.27	92.83	75.13	71.24
DL, α = **40**	95.11	90.27	88.12	67.95	61.11	56.96	93.67	75.93	71.15
VQ, α = **10**	95.38	90.64	88.30	66.43	59.56	55.43	92.31	75.13	70.51
VQ, α = **20**	95.11	90.33	88.31	71.80	62.99	57.73	90.94	75.05	70.26
VQ, α = **30**	95.12	90.09	87.96	70.79	63.13	58.00	93.86	75.66	70.97
VQ, α = **40**	95.31	90.32	88.25	67.78	59.83	55.34	92.04	74.44	69.76

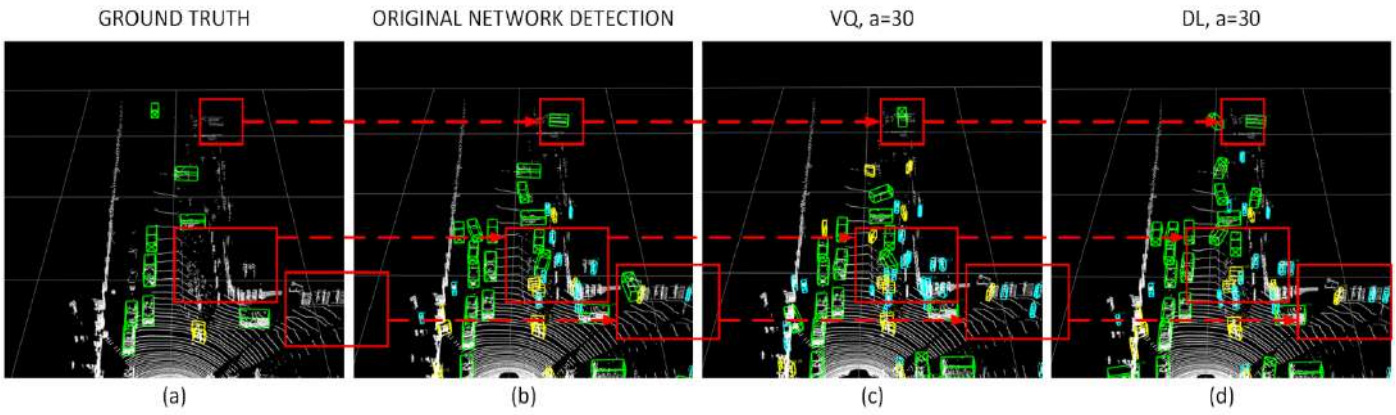


Figure 40: Qualitative example of detection outcomes for Pointpillar. Green, yellow, and blue colors correspond to classes "Car", "Cyclist", and "Pedestrian", respectively. Red box enclosed areas highlight regions of interest for this example.

5 Connection with other Tasks/Deliverables and Conclusion

Work in Task 3.1 is aligned with work in WP2 "Virtual User/Physical Environment Models, CPS Models and orchestration support tools". In particular, the methods for monitoring effectively important aspects of the human activity and behaviour, implemented through the algorithms presented in the present deliverable, are seamlessly integrated in the CPSoSaware simulators in a variety of configurations. All these models are then collected and centrally managed by the Orchestration tools that support autonomic functionality, as delivered in D2.2 "CPSoS orchestration optimization tools" as a result of the work performed in T2.5. Correlation between T3.1 and T2.5 is explained in paragraph 3.2.2.1 of the current report.

Section 2 presented different approaches for user state monitoring. These approaches represent actions on both pillars (i.e., automotive and manufacturing) and they have been developed and used for the purposes of the CPSoSaware project. More specifically, regarding the driver state monitoring, there are three approaches developed by UPAT, CTL and UoP. UPAT has developed a fast and reliable approach for real-time implementation using the dlib library. This DSM approach is proposed to be utilized in scenarios that have been designed to be used with the CARLA simulator. Particularly, the user is monitored and his/her drowsiness is estimated while he/she drives in the CARLA simulator under different traffic conditions (T3.4). The awareness messages that he/she receives depends on both the traffic conditions and his/her drowsiness at the same time (T4.5). CTL has developed a DSM Android application that provides real-time monitoring and assessment of the driver's drowsiness and attention levels. The application uses the smartphone's front camera in order to monitor the general status of the driver during the driving session and to estimate his/her fatigue and distraction levels and it can be used in on-site driving conditions. UoP has developed a DSM application implementation on FPGA for monitoring the driver's drowsiness. It is based on the open-source software package Deformable Shape Tracking (DEST). The FPGA DSM module developed by UoP will be installed in a car and tested using various environmental conditions and drivers. Regarding the operator state monitoring, UPAT has developed algorithms for the landmarks extraction from human postures, using the captured activity of a human from a camera, and evaluated the accuracy and performance in different parameterization settings of the implemented pose estimation approach with direct connections with T2.5 and T4.5. The confidence rate of the operator's pose is also estimated while he/she performs collaborative tasks with the robot in the manufacturing environment, in real-time.

Section 3 described initially four multi-modal scene understanding solutions based on 2D image and 3D point cloud processing. SqueezeDet and ResNet are image-based, while Pointpillar and PVRCNN are point cloud-based. All of them achieved high enough classification accuracy for the task of detecting cars, pedestrians and cyclists using the KITTI dataset. In addition, a multi-modal fusion strategy of has been presented, which effectively combines 2D images and 3D point clouds though the 3D-to-2D geometric projection. Afterwards, three state-of-the-art multi-modal odometry solutions, i.e., ORB-SLAM, DSO and Lego-LOAM have been deployed in the simulated environment of CARLA. For that purpose, their performances have been evaluated under a variety of realistic conditions (day or night, weather

conditions, etc.). All of them achieved very promising results as it is indicated by the different statistical metrics. As a future step, and with direct connection with T4.5, a multi-modal odometry solution which effectively fuses the output of each one of the discussed solutions will be formulated, aiming to offer higher localization accuracy as well as to assess the quality of each individual sensor. For that purpose, the multi-modal fusion strategy focusing on increased scene analysis ability will also be exploited.

Section 4 presented a new clustering-based weight-approximation technique for the acceleration of deep neural networks. The main idea is to utilize a codebook for quantizing network parameters. The codebook contains representative quantities (called codewords, centroids, etc.) for the parameters to be approximated (e.g., vectors of weights). As multiple network parameters are mapped to a single representative quantity, such approaches are also called parameter sharing techniques. Previous approaches proposed vector quantization methods, using the k-means algorithm, for estimating the desired codebook with the aim to compress the weights of fully connected layers. The presented technique exploits the particularities of the problem at hand in order to increase the number of used centroids for the same target acceleration, as compared to the conventional k-means technique. This is achieved using a Dictionary Learning framework, by imposing a special structure to the centroids that reduces the overall computational complexity of the accelerated layer. This section investigates the acceleration benefits of weight sharing methods in image based and LIDAR based deep learning-based scene analysis for automotive CPSs. In the case of image-based object detection, best practices for optimizing the available variables and the training procedures are described based on extensive evaluations on the KITTI dataset. The presented results provide details about the type of errors that manifest, resulting in significant acceleration gains with negligible accuracy losses. By inspecting the error analysis, it can be easily seen that most of the errors are attributed to annotation uncertainties. A more thorough investigation that utilizes also synthetic datasets generated from the CARLA autonomous driving simulator is currently under investigation and it is expected to alleviate the impact of the uncertainties to the training and validation errors, providing additional space for acceleration gains. In the case of LIDAR-based object detection, we investigated the impact of two recently proposed weight sharing MCA techniques on the performance of two state-of-the-art 3D object detectors in the frame of automotive applications, namely Pointpillars and PV-RCNN. The superior performance of the technique was validated via a number of experiments on three well-known state-of-the-art pre-trained DNN models.

The presented acceleration approaches will be utilized to improve the performance on NVIDIA Jetson TX2 and embedded GPU platforms.