D5.2 PRELIMINARY VERSION OF CPSOSAWARE INTEGRATED PLATFORM

Authors	UOP

Work Package	WP5 – CPSoSaware Integration and Cross-layer Optimization supporting design-operation continuum

Abstract

This deliverable is a report that presents the Task 5.2 activities on the Integration and Cross-level Optimizations for CPSoS Maintenance and CPSoS lifecycle Design Operation Continuum Support Management. The activities of T5.2 will be completed by M36 and D5.2 is the "Preliminary Version of CPSoSaware Integrated
Platform". D5.2 presents the CPSoSaware architecture from the integration and interfaces perspective of the technical components (TC). In that context, a set of these components and their integration efforts are given with respect to the use case demonstrations.

Funded by the Horizon 2020 Framework Programme
of the European Union

Deliverable Information

Work Package	WP5 CPSoSaware Integration and Cross-layer Optimization supporting design-operation continuum
Task	T5.2 Integration, Cross-level Optimizations for CPSoS Maintenance and CPSoS lifecycle Design Operation Continuum
Deliverable title	D5.2 Preliminary Version of CPSoSaware Integrated Platform
Dissemination Level	PU
Status	F: Final
Version Number	1.0
Due date	M28

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	25/04/2022	Structure	UOP
0.2	18/05/2022	Draft ready for internal review	UOP
0.3	31/05/2022	Final version of the deliverable	UOP

	NAME
Prepared by	UOP
Reviewed by	ISI, TAU
Authorised by	ISI

DATE	RECIPIENT
	Project Consortium
	European Commission

Table of contents

Figures

Tables

Executive Summary

The scope of this deliverable, *"Preliminary Version of CPSoSaware Integrated Platform"*, is to present the integration and cross level optimization approaches and implementations of the CPSoSaware technical components as defined, described, implemented and reported in previous deliverables ([1] [2]). Since the respective task (T5.2) is still in progress, these activities will be finalized and reported in the next deliverable D5.4 (Final Version of CPSoSaware Integrated Platform).

More specifically in this Deliverable:

- Section 1 describes the objectives and the context of this delivable
- Section 2 gives the overall CPSoSaware architecture with respect to the interfaces and interactions among the components
- Section 3 present a continuous integration and continuous deployment approach and methodology
- Section 4 describes thoroughly the integration efforts performed on scene analysis/understanding and localization components
- Section 5 is devoted to security runtime monitoring and management components
- Section 6 presents hardware acceleration components and how they are demonstrated through stand alone applications
- Section 7 describes the developments conducted on the components related to intra – communications of the CPSoSaware
- Section 8 presents the components related to the AV simulator provided by Robotec and the interface developed for their interaction
- This deliverable concludes in Section 9

1 Introduction

CPSs are designed using a model-based design approach, thus accurate modelling and simulation plays an important role in the design outcome. This approach is similar for CPSoS although we must consider the fact that CPSoS have a continuous evolution that involve the continuous addition, removal, and modification of hardware and software CPS components over the CPSoS complete life cycle. This poses a considerable CPSoS challenge since the CPSoS design phase and operation phase are not separated but rather coexist through time thus forming a design operation continuum that must be supported. This continuum leads to a need for System-wide dynamic reconfigurability and adaptability of CPS resources and CPS process lifecycles. The CPSoS must include a mechanism able to reconfigure its CPS components according to its evolving physical and cyber environment, possibly commission new components or decommission/replace old ones. However, the complexity and autonomy of the CPSoS makes it very hard to identify when a reconfiguration is needed, thus highlighting the need for introducing CPSoS self-awareness through a CPSoS cognitive mechanism. The cognitive CPSoS must be able to provide situational awareness in a decentralized manner (matching the decentralized way CPS operate within the system) and aid both CPSoS operators and users in order to reduce the complexity management burden. CPSoSaware architecture as already presented thoroughly in the respective deliverable ([2],), delivers these requirements through the tight integration of various components that operates in the CPSoS System Layer and CPS/CPHS Layer.

Deliverable D5.2 is the preliminary version of a series of 2 deliverables that describe the integration activities performed in T5.2. T5.2 focuses on the integrations and cross level optimizations for CPSoS Maintenance and CPSoS lifecycle design operation continuum. In this task, the various CPSoSaware blocks that provide support for the CPSoS Design Operation Continuum are integrated and evaluated. This action performs integration of the MRE, the CSAIE, the SRMM and the SAT blocks of the CPSoSaware System Layer with the CP(H)S Layer commissioning, security components, the definitions of the data structures to be exchanged between the CPSoSaware System Layer and the CP(H)S layer and the actual generation of test vectors to be used for the validation of the Design Operation Continuum support mechanism. In this task, the evaluation process that is going to be conducted on the tow use cases in WP6 will be used as feedback in order to provide optimization to the cross – layer integrated components. The evaluation process is meant to highlight possible Requirements KPI misalignments due to integration of the various CPSoSaware blocks and components and provide possible solutions to mitigate the risk. The cross-layer communication will be optimized in order to support the Requirement KPIs, thus focusing on providing fast response time and small communication latency. The evaluation process will also be extended to the level of provided security in the Design Operation Continuum Support Mechanism. The task is associated with all WP5 and WP4 tasks and the evaluation process of WP6.

This deliverable presents a subset of the CPSoSaware components where integration with respect to other components and use cases have been already designed and implemented. This subset comprises of:

- Scene analysis and understanding components
- Localization components
- Security runtime monitoring & management components
- Hardware acceleration components

- Intra communication communication
- AV simulation components

The rest of the components and the holistic integrated CPSoSaware workflow will be reported in the next and final version of this deliverable (D5.4 Final Version of CPSoSaware Integrated Platform).

2 Architecture

CPSoSaware system, as of the latest version of the system architecture, consists of 3 main layers. 1)CPSoS System Layer, 2) CPS/CPHS Layer, 3) Simulation and Training Layer (Figure 1). The distribution of the technical components these layers is presented from Figure 2 to Figure 4. T5.2 is strongly related to T1.3 where the dependencies, interactions and finally interfaces of the various components have been detected. These interactions are depicted in Figure 5. More insights and details on the description/specifications of the system architecture and components is given in "D1.4: Second Version of CPSoSaware System Architecture" from which these figures where excerpted [3].

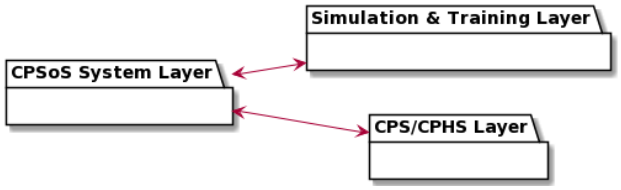


Figure 1 CPSoSaware Layers

Moreover, the outcome of these integration activities as performed in T5.2 and reported in the 2 respective deliverables will be realized in WP6 for the execution of the pilots. During the initial phases of the CPSoSaware project, two use cases have been defined and described in detail. In this definition phase, the use cases are outlined, and the main components have been identified. These developments are to be integrated on the two pilot demonstrators and tested/validated in specific testing scenarios as reported in [3].

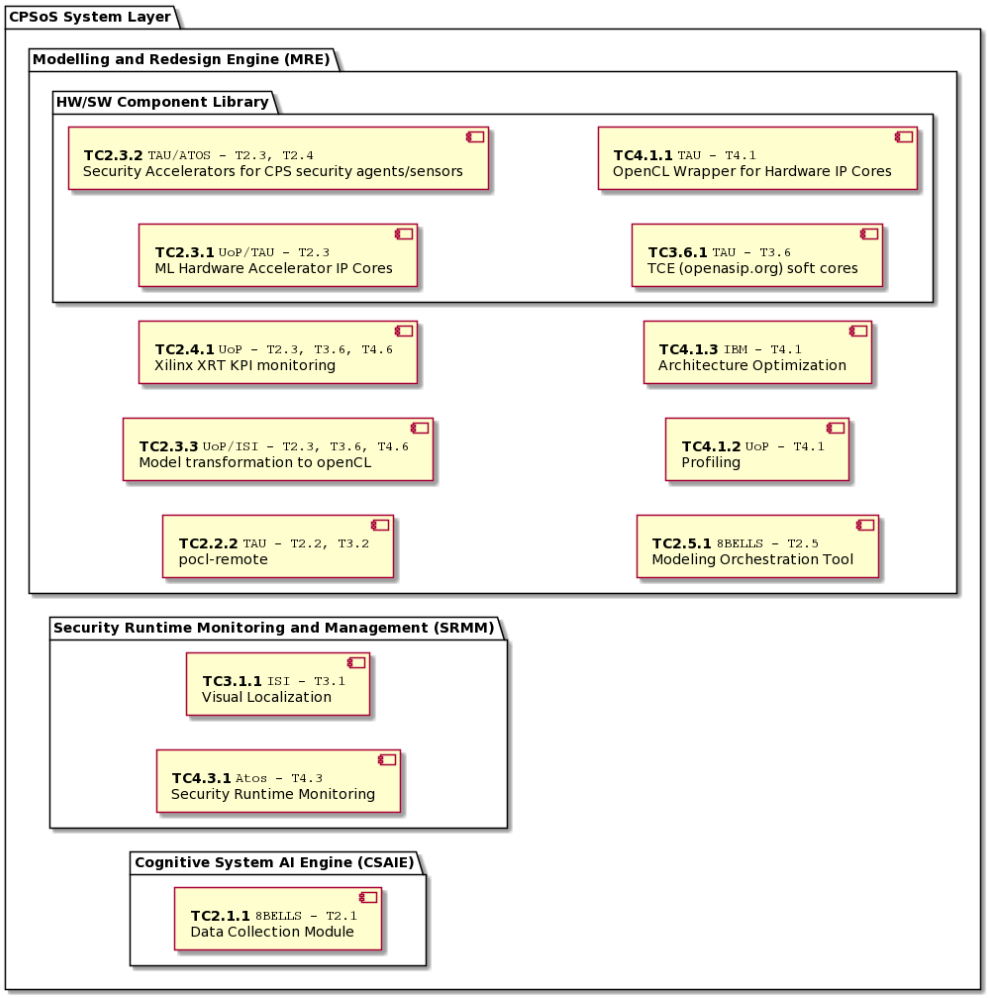


Figure 2 CPSoS layer and sub-blocks

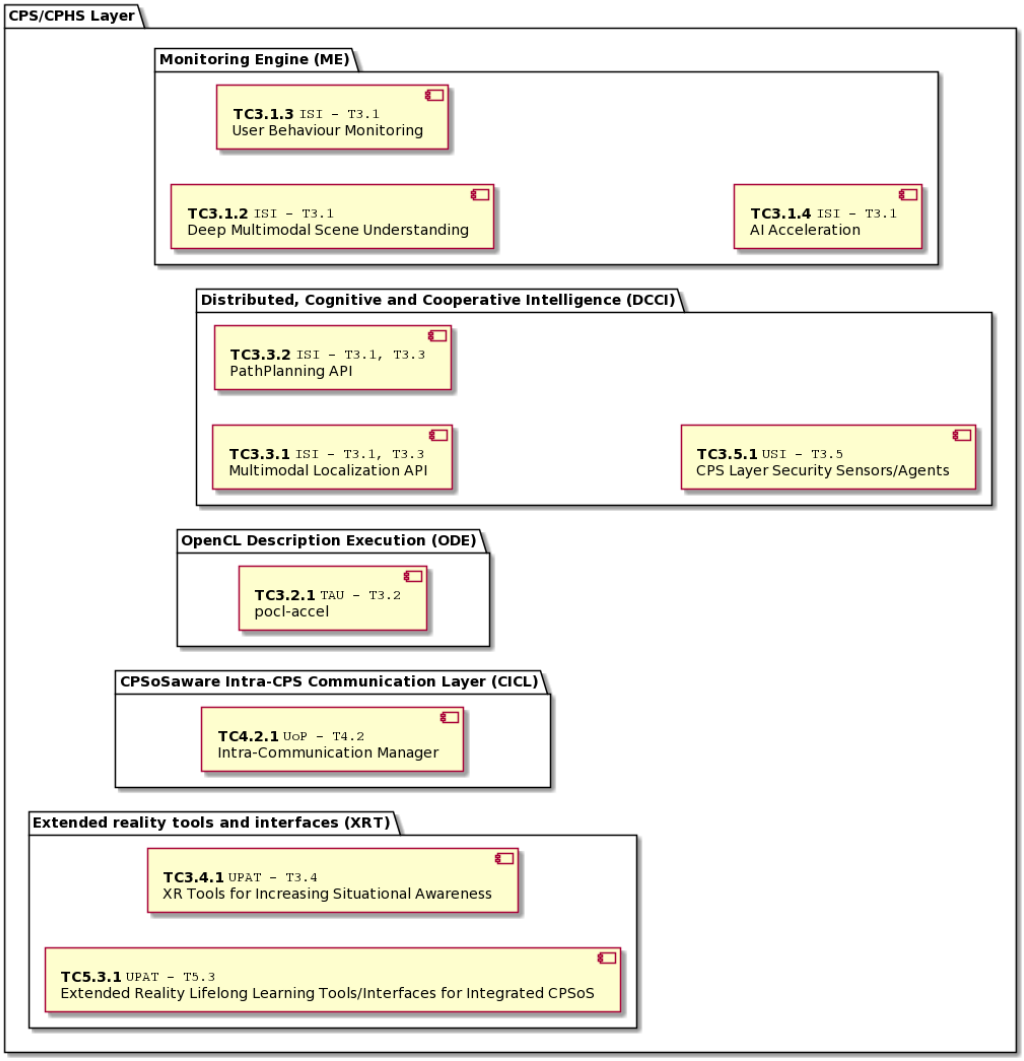


Figure 3 CPS/CPHS layer and sub-blocks

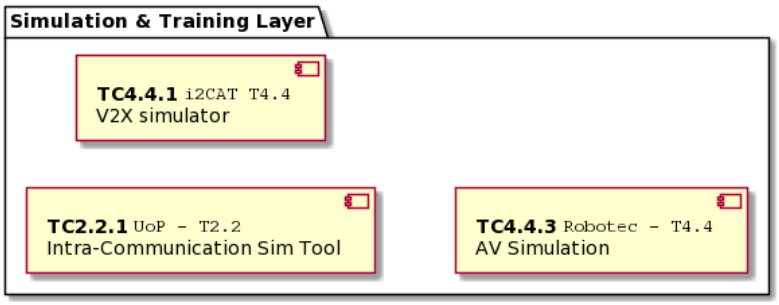


Figure 4 Simulation and Training layer and sub-blocks

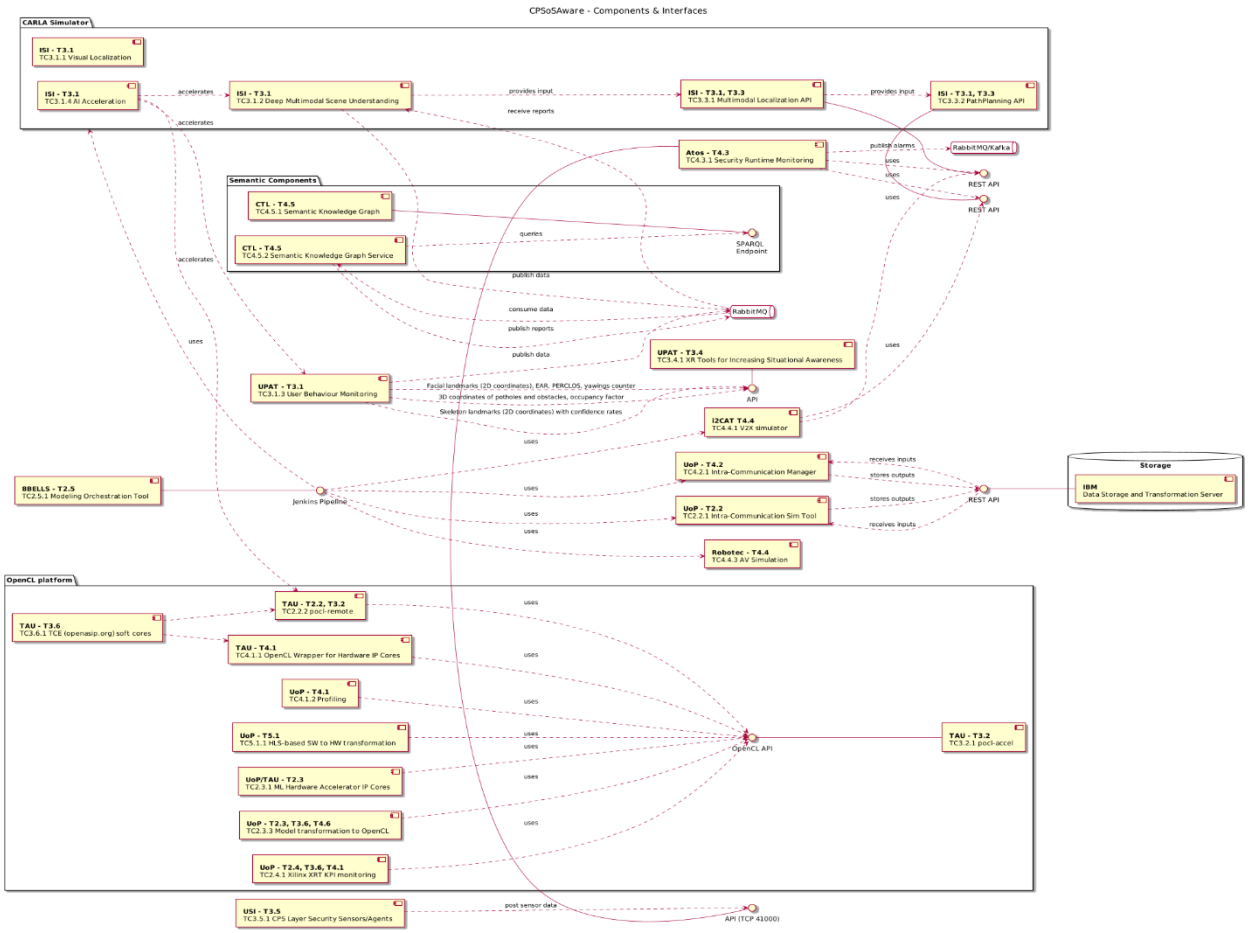


Figure 5 Overview of system interfaces

3 Integration & Deployment Framework

To facilitate a more formal and automated way to perform integration testing and deployment, CPSoSaware adopted the use of Continuous Integration / Continuous Deployment automation servers. In CPSoSaware, automations on integration testing where these are applicable, are based on Jenkins [4], an open source & free software that implements an automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous deployment. It is a server-based system that runs in servlet containers such as Apache Tomcat and it supports several version control tools (e.g. CVS [5], Subversion [6], Git [7], Mercurial [8], etc.) and can execute various build tools commands as well as arbitrary shell scripts and Windows batch commands.

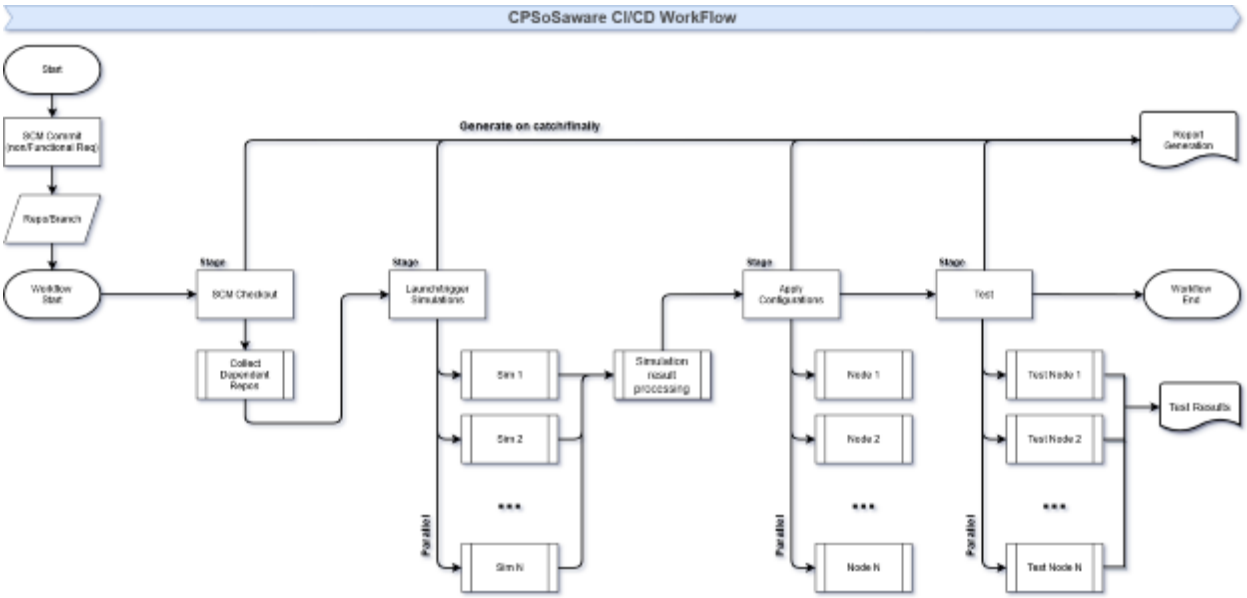


Figure 6: CPSoSaware CI/CD workflow

The workflow proposed in the CPSoSaware project is presented in Figure 6. This workflow is designed based on Jenkins Pipelines [9] and there will be configured with a source code management (SCM) polling trigger.

The SCM system adopted by the CPSoSaware is Git. Git is a distributed version-control system for tracking changes in any set of files, originally designed for coordinating work among programmers cooperating on source code during software development. Its design goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to the users. Every change to the software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment. Pipeline provides an extensible set of tools

for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax. The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository. This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

As already imposed, all the involved components in the CPSoSaware platform will be version controlled and stored in Git Repositories. These components will be:

- Functional/non-Functional requirements
- Simulation suite code
- Components configurations (raspberry, FPGA, etc.)
- Components codes:
  - Bitstreams codes
  - Service codes
  - Scripts
- Test automation scripts: The testing scripts will verify that the configurations are applied/deployed successfully in the components and there is communication between them.

Also, a binary repository manager (also known as artifactory) will be configured to store 3rd party libraries and/or the outcome of the build process. This repository will store binaries such as:

- Customized OS images
- FPGA bitstreams
- Simulation suite binaries

It must be noted, CPSoSaware components present a heterogeneity that does not allow in the context of the project, to configure pipelines where end – to – end workflows will be able to be automated through the CI/CD automation server. However, individual integration paths have been already tested through Jenkins pipelines while automated deployment/commissioning tasks are to be executed by Jenkins delivering the required functionality of the TC4.61 as described in D1.4 [2]. The details of the automation server maintained from UOP along with the storage and transformation (SAT) engine developed from IBM that is used for persisting configuration data and evaluation results, are detailed in more details in D4.4 [10].

4 Scene Analysis & Localization Components

In the following chapters we provide a short description of the algorithms examined and the solutions developed by ISI. More specifically in Chapter 4.1 we present scene analysis algorithms and in Chapter 4.2 localization algorithms. In Chapter 4.4 we present how we integrated these solutions in the Carla ROS framework, and we provide the relevant repository links and technical instructions. Moreover, in the same chapter we present the integration of the Carla Artery simulator to ROS framework, something which gives us the ability to simulate more realistically the V2X communications. In addition, in the same chapter we present the integration of the visual odometry algorithms to a real vehicle. In Chapter 4.5 we give a description of the framework and the REST API developed for publishing the framework's resources. The metrics used for the quantitative evaluation of the presented solutions are described in Chapter 4.6. Finally, in Chapter 4.7 we present the integrated demo setup and the workflows that we're going to implement.

4.1 Scene analysis understanding accelerated modules

In the following chapter we present a short description of the scene analysis algorithms that have been tested. You can find more information about the algorithms in Deliverable 3.1 [11]. The following solutions and the relevant integrations described implement components **TC3.1.2R1 - TC3.1.2R3.**

The version of multimodal 2D and 3D scene analysis with acceleration which can be executed in a PC or an embedded system (Jetson) can be found in the following links:

- https://gitlab.com/isi_athena_rc/cpsosaware/multimodal-scene-understanding/multimodal-fusion-tools
- https://gitlab.com/isi_athena_rc/cpsosaware/multimodal-scene-understanding/multimodal-fusion-tools/-/tree/jetson

4.1.1 2D image-based scene analysis

We integrated and evaluated the performance of the VQ and DL weight sharing techniques on the two fully convolutional object detection networks, namely, SqueezeDet and ResNetDet, presented in [12]. Both networks utilize a feature-extraction part that translates the input image into a high-dimensional feature map, followed by ConvDet, namely a convolutiona detection layer with the purpose of locating object-containing bounding boxes, predict the class of each object, and produce a confindence score for each detection. Where the two networks differ primarily is the in feature-extraction part, with SqueezeDet utilizing SqueezeNet [13] as a backbone network while ResNetDet being based on ResNet50 [14].

4.1.1.1 Accelerated models

We apply the 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 [15] 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.

Integrating the accelerated version corresponds to the utilization of weights that have been accordingly processed with the proposed in D3.1 [11] weight sharing approaches.

4.1.2 3D point cloud based scene analysis and understanding

Here, the two object detection schemes that will be considered, namely, PointPillars and PV-RCNN, are briefly presented. The PointPillars network [13] 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 [16] and Second [17], by utilizing strictly 2D convolutions, thus, achieving both high precision and fast inference.

4.1.2.1 Accelerated models

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.

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 Fig. 2.3(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 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.

The main bulk of the operations required by PV-RCNN are consumed by the Voxel-Backbone and the BEV-Backbone blocks shown in Fig. 2.3(b), with the former one being composed of Submanifold Sparse 3D-Conv layers [18], 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 Fig. 2.3(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 Fig. 2.3(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 ×, 5.5 ×, 6.0 ×, and 6.3 ×, respectively

Likewise , Integrating the accelerated version corresponds to the utilization of weights that have been accordingly processed with the proposed in D3.1 [11] weight sharing approaches.

4.1.3 Multimodal fusion

A late fusion strategy takes place combining 2D driven detections and 3D driven detections. Initially, 3D bounding boxes are projected upon the 2D plane and converted to 2D bounding boxes. To fuse 2D and 3D measurements a non-maximal suppression [19] driven approach takes place redefining the bounding boxes on the 2D space. Afterwards, to define vehicle range measurements 2D projects are matched to 3D points of the point cloud. Subsequently, each 3D point of the point cloud [x_i, y_i, z_i] is projected upon the 3D image.

The 3D bounding box is described by its center $T = [t_x, t_y, t_z]^T$, dimensions $D = [d_x, d_y, d_z]$, and orientation $R(\theta, \phi, \alpha)$ where θ is the azimuth, ϕ is the elevation and α is the roll angles. Given the pose of the object in the camera coordinate frame $(R, T) \in SE(3)$ and the camera intrinsics matrix $R_{Rect}^{(0)}$, the transformation matrix $P_{Rect}^{(i)}$ the projection of a 3D point $X_o = [X, Y, Z, 1]^T$ in the object's coordinate frame into the image $x = [x, y, 1]^T$ is:

$$x = P_{Rect}^{(i)} * R_{Rect}^{(0)} * X$$

Detection networks assign an objectness score to each anchor-box. For an image of size $W \times H$ pixels, a total of $\frac{W}{S} \times \frac{H}{S}$ anchor-placement locations are obtained, where S is the stride of the CNN. Given an anchor-template set $\mathcal{T} = \{T^1, T^2, \dots T^k \dots T^K\}$, corresponding to different scales and aspect-ratios, a total of $K \times \frac{W}{S} \times \frac{H}{S}$ anchor-boxes, $\mathcal{A}$, are placed over the image. Every element in $\mathcal{A}$ is assigned an objectness score. Therefore, for a 1024×1024 pixels image, a CNN of stride 16 and 15 anchors per location, generates a total of 61,440 proposals forming the set $\mathcal{S}$. NMS is applied to remove spatially redundant proposals that are very close to each others while ensuring high recall for all the objects in the image with a limited candidate set.

The popular NMS algorithm is sequential in nature. At each iteration i, it selects the top scoring proposal $P(i)$ from the set $\mathcal{S}$ and removes all proposals in $\mathcal{S} - P(i)$ which have an overlap o greater than a threshold t. The complexity of each iteration is linear in the size of set $\mathcal{S}$.

Algorithm 1: Non-Maximum Suppression

Input: B^c, S^c, N_t
Initialisation:
$D^c \leftarrow \{\}$
while $B^c \neq empty$ **do**
  $m \leftarrow \text{argmax}\, S^c$
  $M \leftarrow b_m^c$
  $D^c \leftarrow D^c \cup M$
  $B^c \leftarrow B^c - M$
  for $b_i^c \in B^c$ **do**
    if $IoU(M, b_i^c) \geq N_t$ **then**
      $B^c \leftarrow B^c - b_i^c$
      $S^c \leftarrow S^c - s_i^c$
Output: D^c, S^c.

Figure 7 Non maximum suppression algorithm.

4.2 Odometers

In the following chapter we present a short description of the odometry algorithms that have been tested. You can find more information about the algorithms in Deliverable 3.1 [11]. The following solutions and the relevant integrations described implement components **TC3.1.1** and **TC3.1.2**.

4.2.1 DSO

Direct Sparse Odometry is a Visual odometry method which exploits a probabilistic model by minimizing photometric error with consistent, joint optimization of all model parameters, including geometry-represented as inverse depth in a reference frame-and camera motion. 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.

The version of DSO which can be executed in a PC or an embedded system (Jetson) can be found in the following link:

https://gitlab.com/isi_athena_rc/cpsosaware/odometers/dso

4.2.2 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 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.

LeGO Loam can be found in the following link.

https://gitlab.com/isi_athena_rc/cpsosaware/odometers/lego_loam_ros

4.2.3 ORB SLAM 2

ORB-SLAM 2, is one of the most popular open-source feature-based monocular SLAM systems that can be executed in real time. It outputs an estimated camera trajectory and a sparse point cloud reconstruction of the environment. The system is robust to severe motion clutter, allows wide baseline loop closing and relocalization, and includes fully automatic initialization. ORB SLAM 2 executes in parallel three distinct processes which implement the following tasks: Tracking, Local Mapping and Loop Closing.

https://github.com/raulmur/ORB_SLAM2

4.2.4 Multi-modal relocalization

In terms of the odometry robustification and assessment in a variety of testing scenarios for the purposes of T3.6 and T4.5, we have derived the proposed system architecture diagram of Figure 8, through the combination of a Visual and LIDAR based SLAM solution:

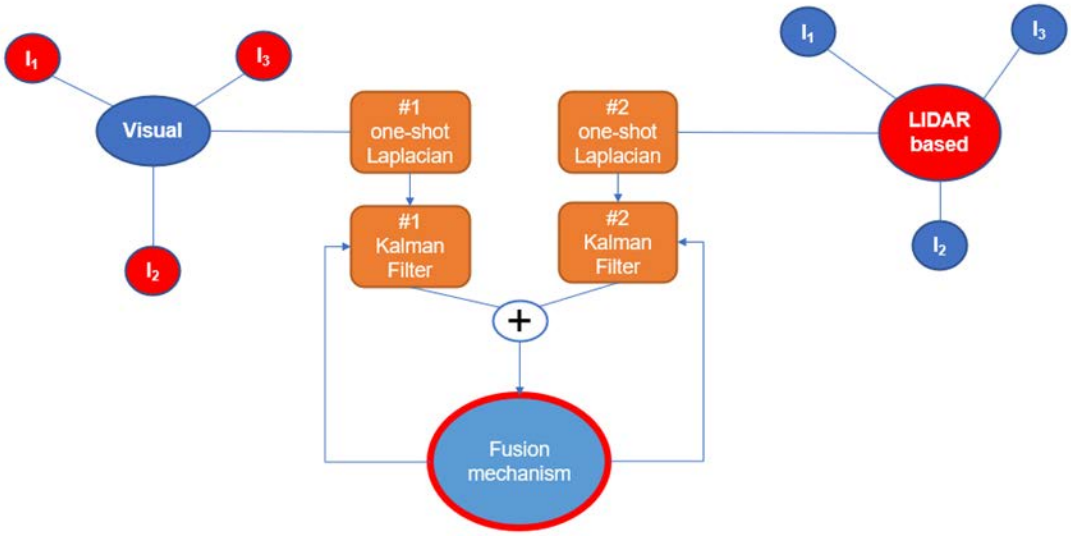


Figure 8 Proposed multi-modal fusion architecture

Its core idea is to couple the extracted pose from Visual (LIDAR based) approach with the landmarks detected by the LIDAR based (Visual) solution.

4.3 Cooperative Localization

For the Cooperative Localization, Tracking and Awareness of T3.3, we have assumed that vehicles in an urban environment through V2V communication exchange their measurements so as to estimate more accurately their positions. For that purpose, a graph based approach has been utilised, which couples together the vehicles' connectivity through the graph Laplacian operator, with the multi-modal inter-vehicular measurements. Two general methodologies have been developed: (i) information diffusion based, in which neighbouring vehicles broadcast and receive in an iterative manner the estimated location vectors, (ii) Kalman Filter based, where the state vector that needs to be tracked contains the self and neighbouring vehicles' positions. The latter approach is able to address the dynamic nature of connectivity topologies highly efficiently. In both cases, the key step of extracting the range measurements (relative distance and angle) towards other vehicles has been skipped. Our goal is to explicitly integrate the previously discussed object detectors in the specific framework of Cooperative Awareness, so as to employ realistic traffic data.

Another limitation of the developed methodology is related to the apparent network delays in vehicular applications. Since vehicles frequently exchange measurements and estimations, it is expected that the delay introduced by V2V communication impacts on the performance of Cooperative Awareness. However, we have realistically simulated the effect of delays in our framework, so as to initially evaluate their footprints. Each vehicle broadcasts CAM messages at least every 100 ms, while the maximum delay introduced by V2V communication can reach 300 ms at heavy traffic density of 0.1 vehicles/meter. Therefore, for every iteration round of the proposed diffusion-based algorithm we have at most 400 ms delay, which implies that vehicle i receives the location vector of its neighbors estimated 4 iterations before. Integrating a

network and communications simulator in the environment of CARLA, alongside the object detectors will be another major goal of ours.

4.4 Levels of integration

In this chapter we present the different levels of integration regarding the components that implement the algorithms described in the previous chapters. In the first level the components are integrated in the Carla-ROS simulation framework, in the second level we have the addition of V2X communication due to the integration of Carla Artery simulator to ROS framework, and finally, in the third level we have the deployment of the components in a real vehicle provided by Panasonic.

4.4.1 CARLA ROS

The integration to the Carla-ROS framework is the development of the appropriate ROS nodes that will implement a specific algorithmic behavior. These nodes either contain all the resources necessary for executing the algorithm or the call an appropriate library.

Every ROS node consumes and published data in the context of the ROS framework, under ROS topics. The type of these messages is either predefined by the ROS framework or custom types can be created and used. The synchronization of all the nodes is provided by the ROS framework. Concerning the programming languages, the nodes are either implemented in Python or in C++.

All the nodes consume data generated in the Carla simulation environment. The data can be either generated asynchronously (rosbags) or synchronously. In the latter case, the user of the carla-ros-bridge is necessary for the establishment of the bi-directional communication between Carla and the ROS framework.

4.4.1.1 DSO

The DSO ROS node is a ROS wrapper of DSO. It subscribes to a topic under which the rgb image data are published, it sends the data to DSO, it gets back the estimated pose and the DSO points, and it published them under the relevant topics.

Requirements

- DSO installed
- Pangolin
- OpenCV

Installation and execution

1. Install DSO
2. Download the repository in a catkin workspace

```
git clone https://gitlab.com/isi_athena_rc/cpsosaware/odometers/dso_ros.git
```

3. Build:

```
export DSO_PATH=[PATH_TO_DSO]/dso
```

```
catkin_make
```

4. Run

```
rosrun ros_dso ros_dso ros_dso_node image:= /ego_vehicle/rgb_fron/image calib=<path to camera calibration file>
```

Calibration File

The calibration file is used to define the intrinsic parameters of the camera sensor. In the case of Carla we deploy a camera based on the pinhole model and the relative calibration file is based on the following template.

```
Pinhole fx fy cx cy

in_width in_height

"crop" / "full" / "none" / "fx fy cx cy 0"

out_width out_height
```

The parameters fx, fy, cx, and cy denote fx fy cx cy denote the focal length and the principal point relative to the image width and height

ROS Topics

Topic	Type	Description	Message Type
/carla/ego_vehicle/rgb_front/image	Input	RGB input	sensor_msg::Image
/carla/ego_vehicle/state	Input	The state of the ego vehicle (Calibration, Running, Finished)	std_msgs::String
/dso/pose	Output	The estimated pose in reference to dso/odo	geometry_msgs::PoseStamped
/dso/pose_map	Output	The estimated pose in reference to map	geometry_msgs::PoseStamped
/dso/intitial_pose	Output	The pose at dso/odo	geometry_msgs::PoseStamped
/dso/path	Output	The path of ego_vehicle	nav_msgs::Path

/dso/pointCloud	Output	The mapped points	sensor_msgs::Point-Cloud2
/dso/error	Output	ATE error	std_msgs::Float32

Transforms

Transform	Type	Description
carla/ego_vehicle/rgb	Input	Transform for calculating the ATE error
dso/odo	Output	Initial transform at frame 0
dso/cam	Output	Estimated pose

4.4.1.2 ORB SLAM2

ORB SLAM 2 executes in parallel three different processes and more specifically Tracking, Local Mapping and Loop Closing.

Requirements

Eigen3

ROS Topics

Topic	Type	Description	Message Type
/carla/ego_vehicle/rgb_front/image	Input	RGB input	sensor_msg::Image
/carla/ego_vehicle/rgb_front/cam-era_info	Input	RGB sensor in-formation for calibrating the camera	sensor_msgs::CameraInfo
/dso/pose	Output	The estimated pose in refer-ence to the odometry origin	geometry_msgs::PoseStamped

/orb/path	Output	The path of ego_vehicle	nav_msgs::Path
/orb/map_points	Output	The mapped points	sensor_msgs::PointCloud2

Transforms

Transform	Type	Description
carla/ego_vehicle/rgb	Input	Transform for calculating the ATE error
orb/odo	Output	Initial transform at frame 0
orb/pose	Output	Estimated pose

4.4.1.3 LeGO-LOAM

In contrast to DSO and similarly to ORB SLAM2 all the functionalities of LeGO-LOAM are implemented in the context of ROS framework.

Requirements

Georgia Tech Smoothing and Mapping library (gtsam) 4.0.0-alpha2. Execute the following procedure to install the library.

```
wget -O ~/Downloads/gtsam.zip \ https://github.com/borglab/gtsam/archive/4.0.0-alpha2.zip


cd ~/Downloads/ && unzip gtsam.zip -d ~/Downloads/
cd ~/Downloads/gtsam-4.0.0-alpha2/


mkdir build && cd build
cmake ..
sudo make install
```

Installation and execution

```
cd ~/catkin_ws/src
```

```
git clone \ https://gitlab.com/isi_athena_rc/cpsosaware/odometers/lego_loam_ros
cd ..
catkin_make -j1
```

If the code is built correctly, you can run the application.

```
roslaunch lego_loam run.launch
```

ROS Topics

Topic	Type	Description	Message Type
carla/ego_vehicle/lidar	Input	Ego vehicle's lidar (16 or 64 channels)	sensor_msg::PointCloud2
carla/ego_vehicle/state	Input	The state of the ego vehicle (Calibration, Running, Finished)	std_msgs::String
/segmented_cloud	Output	The estimated pose in reference to dso/odo	sensor_msgs::PointCloud2
/laser_cloud_sharp	Output	Sharp features	sensor_msgs::PointCloud2
/laser_cloud_less_sharp	Output	Less sharp features	sensor_msgs::PointCloud2
/laser_cloud_flat	Output	Flat features	sensor_msgs::PointCloud2
/laser_cloud_less_flat	Output	Less flat features	sensor_msgs::PointCloud2
/laser_odom_to_init	Output	Estimated poses	sensor_msgs::PointCloud2
/ground_cloud	Output	The ground plane of the point cloud	sensor_msgs::PointCloud2

Transforms

Transform	Type	Description

map	Input	World frame
camera_init	Output	Odometry coordination frame
camera	Output	Estimated pose frame

Figure 9 Screenshot of real-time ATE and RPE error estimation during simultaneous execution of DSO and LeGO LOAM

4.4.1.4 2D object detection

The following ROS node uses a pretrained YOLO3 CNN for the detection of objects in images.

Requirements

- OpenCV
- boost

Installation and execution

Navigate to the src folder of your catkin workspace and clone the repository.

```
git clone git@gitlab.com:isi_athena_rc/cpsosaware/multimodal-scene-understanding/2d-image-based/darknet_ros.git
```

Build the workspace

```
cd ..

catkin_make -DCMAKE_BUILD_TYPE=Release
```

Start the ROS node

```
roslaunch darknet_ros darknet_ros.launch
```

ROS Topics

Topic	Type	Description	Message Type
/carla/ego_vehicle/rgb_front/image	Input	RGB input	sensor_msg::Image
/object_detector	Output	Number of detected objects	std_msgs::Int8
/bounding_boxes	Output	A custom message that gives information about the position and the size of the bounding boxes in pixel coordinates	darknet_ros_msgs::BoundingBoxes
/detection_image	Output	The image including the bounding boxes	sensor_msgs::Image

The darknet_ros_msgs::BoundingBoxes is a special type of message which except from a header contains an area of the class darknet_ros_msgs::BoundingBox. The message darknet_ros_msgs:: BoundingBox encapsulates the following fields:

```
float64 probability
int64 xmin
int64 ymin
int64 xmax
int64 ymax
int16 id
string Class
```

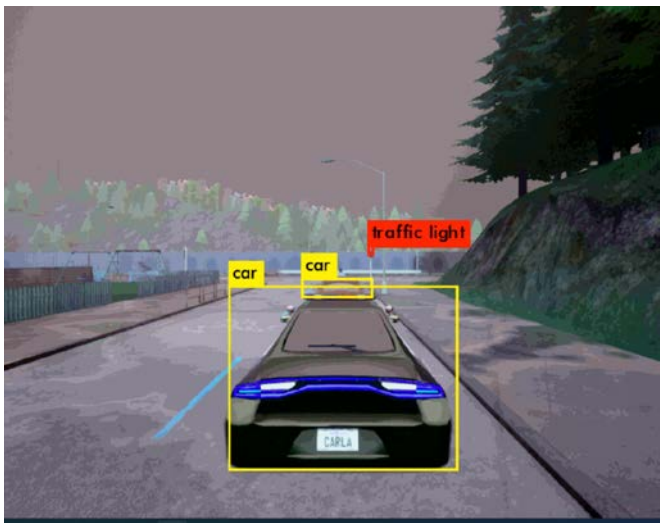


Figure 10 Screenshot of Darknet detecting objects on images from Carla

4.4.1.5 3D object detection

The 3D object detection ROS node used OpenPCDet for the detection of the objects inside the lidar point cloud. It is actuall a ROS wrapper that calls loads the weights of the accelareted models described in the previous chapter.

Requirents

The ROS node has the same dependencies with OpenPCDet except from the ROS framework

Installation and execution

Navigate to the src file of your catkin workspace and download the repo

```
cd ~/catkin_ws/src
git clone git@gitlab.com:isi_athena_rc/cpsosaware/multimodal-scene-understanding/openpcdet-ros.git
```

Build the workspace

```
cd ..
catkin_make -DCMAKE_BUILD_TYPE=Release
```

Start the ROS node

```
roslaunch openpcdet 3d_object_detector.launch
```

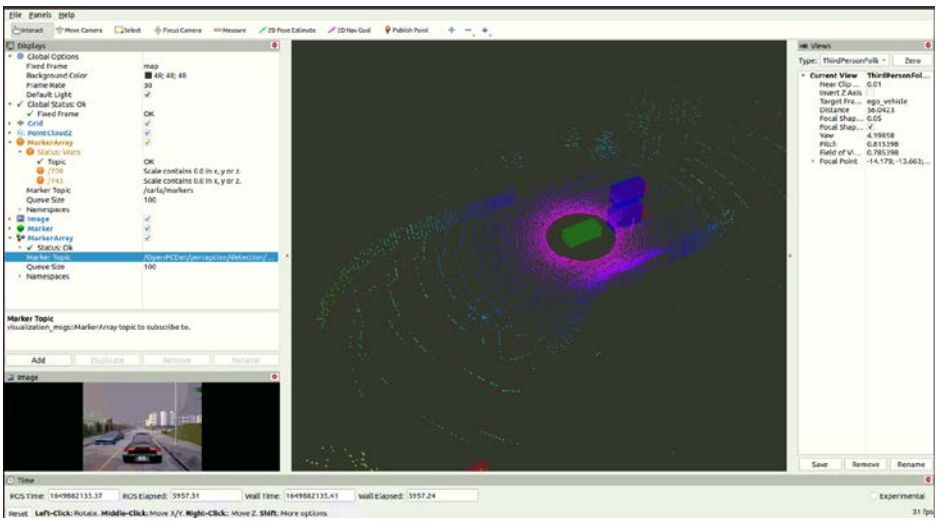


Figure 11 Screenshot of rviz depicting the ego vehicle (green box), estimated 3d objects (blue boxes), ground truth (red boxes) and the point cloud..

Configuration

Navigate inside the src/inference.py file of the repository and change the line that points to the model in line 414. In that way you can use a different model that will execute the inference.

ROS Topics

Topic	Type	Description	Message Type
carla/ego_vehicle/lidar	Input	Ego vehicle's lidar (16 or 64 channels)	sensor_msg::PointCloud2
OpenPCDet/perception/detection **/3D_lidar_obstacles_markers**	Output	An Array of Markers for each 3D object detected in space	visualization_msgs::MarkerArray.msg

4.4.1.6 Cooperative localization

The Cooperative localization ROS Node implements the algorithm of cooperative localization as described in the previous relevant chapter. The repo contains a rosbag file for testing purposes.

Requirements

- numpy
- scipy

Installation and execution

Navigate to the src file of your catkin workspace and download the repo

```
cd ~/catkin_ws/src
git clone https://gitlab.com/isi_athena_rc/cpsosaware/cooperative-localization-and-tracking/ros_ekf
```

Build the workspace

```
cd ..
catkin_make -DCMAKE_BUILD_TYPE=Release
```

Start a ROS core

```
roscore
```

Start the cooperative localization node

```
rosrun extended_kalman node.py
```

Provide data by either utilizing the **rosbag** provided

```
rosbag play ~/ros-workspace/ros_ekf/rosbags/10m.bag
```

or by **using Carla and carla ros-bridge.**
ROS topics

Topic	Type	Description	Message Type
ekf/neighbors	Input	Information about the neighboring behicles	NeighborList
edk/pose	Output	The pose of the ego vehicle	geometry_msgs::PoseStamped
ekf/path	Output	An array of poses of the ego vehicle	nav_msgs::Path

The NeighborList type of message is custom type of message which contains an array of messages of the custom type NeighborInfo, which contains the following fields.

```
uint32 id
float32 x
```

float32 y float32 ate_error float32 gps_error

This type of message is replaced with messages of type ros-etsi-its-messages which follows the official ETSI ITS Cooperative Awareness Message standard. In the following chapter we describe in more detail the integration of the Carla Artery simulator to the ROS framework something which gives us the ability to integrate V2X CAM messages in our scenarios.

4.4.1.7 Logger

The Logger is an auxiliary ROS node that executes the following operations:

- It gathers data by subscribing to ROS topics
- It visualized data or metrics
- It forwards the data to other entities (storage service)
- It stores data to csv files

4.4.2 Carla ROS Artery Simulator

The Carla ROS Artery integrated simulator refers to the combination and synchronization of the different clocks of the various sub-systems of the combined simulation framework. The framework combines three sub-systems, a network simulator, a traffic simulator and a game engine-based simulator, into a single platform. More specifically, CARLA is the component responsible for simulating physics phenomena and rendering. Artery V2X Simulation framework, which is built on top of OMNET++ framework, is for simulating network communications and more specifically V2X communications and SUMO for simulating complex traffic scenarios.

CARLA interacts with ROS through the Carla-ROS Bridge. Since in synchronous mode, only one client can tick the CARLA server, the Bridge must be also launched in passive mode, for the timing of the ROS subsystem to follow the single system clock source, too. Finally, to in order to export to the ROS subsystem important application-level information, such as the ETSI ITS CAM or neighborhood from the Artery/OMNET++ network simulation, the ros-etsi-its-messages [20] encapsulation library can be used. For this, the artery CA service has been instrumented to provide an efficient dissemination of the current snapshot of the constructed neighborhood table related to each CARLA vehicle ID, as built from its own ITS CAM process, which is the ultimate abstraction needed at the ROS application code level.

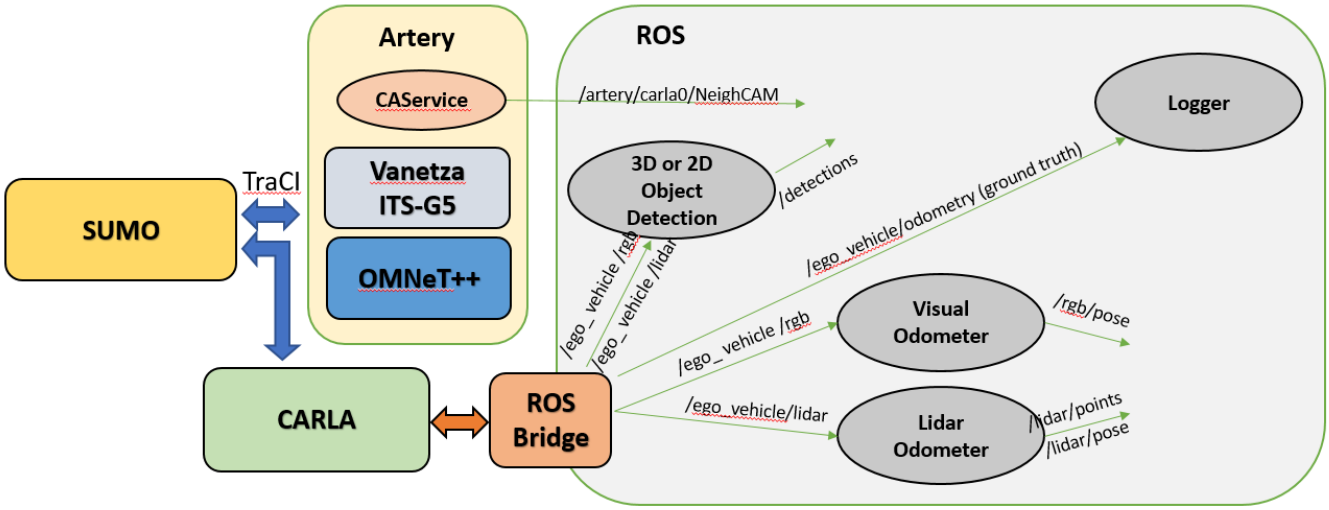


Figure 12 Integrated Simulator's architecture

4.4.3 Integration in PANA's vehicle of odometry solutions

The basic task of this module is to choose and fuse inputs from the odometry sources offered by our system in order to perform vehicle localization as robust as possible.

4.4.3.1 Integration of GPS Information

Since GPS is the only available input that allows for an absolute positional measurement, it seems natural that GPS information should be included in the odometry fusion. All other odometry solutions follow the dead reckoning principle, meaning that they have no means to recover from any inaccuracies in a past frame. However, the frame-to-frame accuracy of GPS is much lower than of the other means of collecting odometry information. Experiments indicate that this coarseness makes a direct integration of GPS information in real-time odometry very difficult if the excellent detail of the odometries should be retained. The first approach of using GPS information in odometry fusion thus is not a real-times one but one that alters that odometry of past frames as well. While this has serious drawbacks for application use, it shows that an integration of GPS is possible while retaining smooth trajectories (see 4.4.3.4).

4.4.3.2 Dependencies

During testing and development, the odometry fusion module certainly depends on all modules which create an odometry as their output. Once the work in this module has become mature enough, it may be possible to cancel some of these dependencies if the odometries are not needed as input.

Apart from the common algorithm and math library, odometry fusion depends on the modules that provide:

- Vehicle Odometry

- Visual Odometry
- Dense Slam
- Sparse Slam
- GPS

4.4.3.3 Third-Party Dependencies

OpenCV is used only during debugging to store images etc. It is not part of the implementation of any of the algorithms. No other third-party libraries are being used.

4.4.3.4 Description of the Architecture

In order to find the best possible solution, odomfusion is being tested with several different algorithms.

4.4.3.4.1 Real time Fusion with Kalman Filter

Fusing of multimodal odometries

The original implementation uses an Extended Kalman Filter to fuse visual and vehicle odometry. It has been updated to exhibit the same interface as the other estimators, but the algorithm remains unchanged. The inputs are hard-coded so there is no straight-forward way to extend this filter.

Multi input Kalman

A new estimator has been introduced which contains an extensible Kalman Filter and allows for any number of different inputs, which must be decided at compile time because of the fixed matrix sizes used in all computations. At the moment, the algorithm uses a simple linear Kalman Filter on the car position only but is easily extensible to accommodate nonlinear functions. Figure 13 shows a comparison between the original EKF implementation, which fuses vehicle and visual odometry, and the multi-input linear KF fusing odometries derived by multiple modalities. All base estimators yield quite differing results for the chosen test sequence, and it is visible that the addition of visual slam input improves the result, as the optimal result would be ending exactly where the track started.

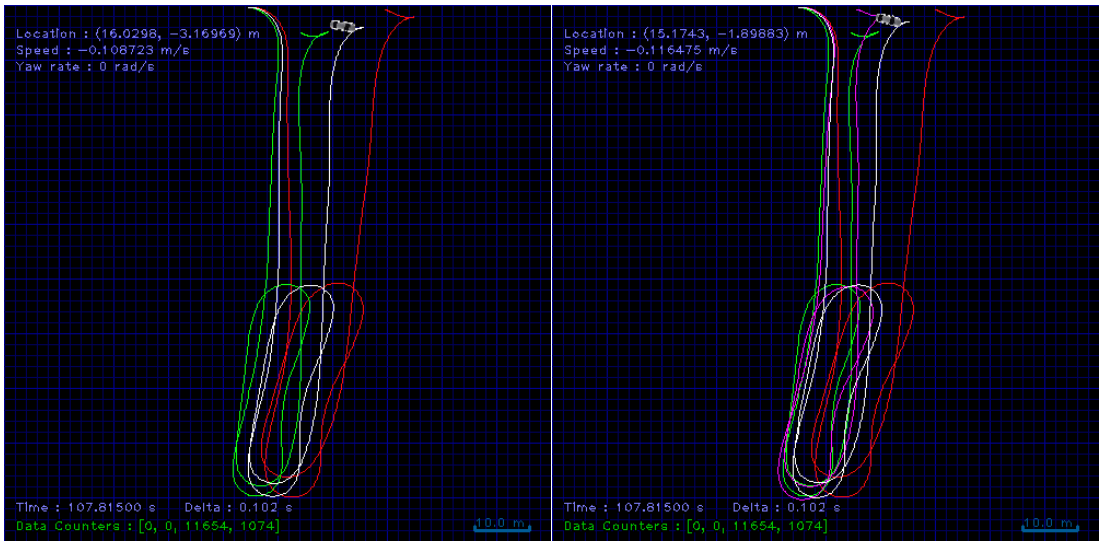


Figure 13 (a) original EKF. (b) Linear EKF. Comparison of different fusion algorithms only a track where the base odometries yield differing results. White is the fused odometry, while the others are: red - vehicle, green - visual, pink – Visual SLAM

4.4.3.5 Non-Real Time Fusion Using Nonlinear Optimization

The first estimator to include GPS information uses nonlinear optimization over a window of past frames. In each frame, it computes the affine transformation that minimizes the distance to the GPS trajectory if applied to the trajectory points in a windowed fashion with its weight declining over time.

4.4.3.6 Offline Parameter Optimization

Since we have several combinations of filters as well as input configurations at our disposal (e.g: EKF with veh/gps/vis odometries, or LKF with veh/vis/gps/ Visual Dense Slam), a method for comparing these setups is needed. Furthermore, since all of these setups have a number of parameters which can possibly be used to improve fusion results, it is desirable to have a principled method of assessing the fusion quality provided by a set of parameters for each setup. Additionally, a method allowing to quantitatively compare the setups and parameter settings with each other would allow for an automatic approach to find the best combination of estimation setup and parameters.

4.4.3.7 Viewing Odometry Fusion as an Optimization Problem

With the possibility to obtain an error - or more precisely a residual vector when using multiple datasets, we have all ingredients to view the whole odometry fusion module as an optimization problem. A certain setup gives rise to certain parameters (e.g., variances and covariances for process and measurement noise in a Kalman Filter) for which a residual vector can be evaluated using the offline optimization tool. When changing the parameters, we can check if we can achieve a lower error by taking some norm of the residual vector. If we do this in a systematic fashion, we are mathematically optimizing over the parameters for the chosen odometry fusion setup. While traditional optimizers like Gauss-Newton have failed due to the complex error function and its potentially unreliable derivatives as calculated using _nite differences.

However, we can use a derivative-free optimization algorithm which is more robust against problematic error functions. However, these optimizers take many more evaluations of the error function in order to converge or find a region with low errors than those guided by derivatives. But in the case of optimizing the odometry fusion parameters, this is not a problem because

The offline tool is implemented to allow extremely fast evaluation of the whole odometry fusion process to build up the residual vector for a certain set of parameters

The process is not time-critical since it is part of the development for the odometry fusion module, and does not need at all to be used in the real-time system

Figure 14 illustrates the potential of this optimization for two different datasets which both have the same start and end location. The input trajectories are shown in red (vehicle odometry), green (visual odometry), and pink (Visual Dense Slam). The setup for odometry fusion is an Extended Kalman Filter (EKF) which fuses the vehicle odometry and Visual Dense Slam inputs. The trajectory calculated by the fusion is shown in white. The left image in each row shows the fusion as computed by the initial (hand-picked) parameters that have been used so far, where the right image shows the trajectory as computed by the same fusion algorithm, using the optimized parameters.

4.4.3.8 Integration of Optimized Parameters

In order to make use of the optimized parameter settings and also to ease updating them in the future (e.g. if more datasets are available or new methods for calculating the quality arise), they have been placed in the config file for odometry fusion which has been introduced for this cause. The fusion algorithm itself can easily be changed.

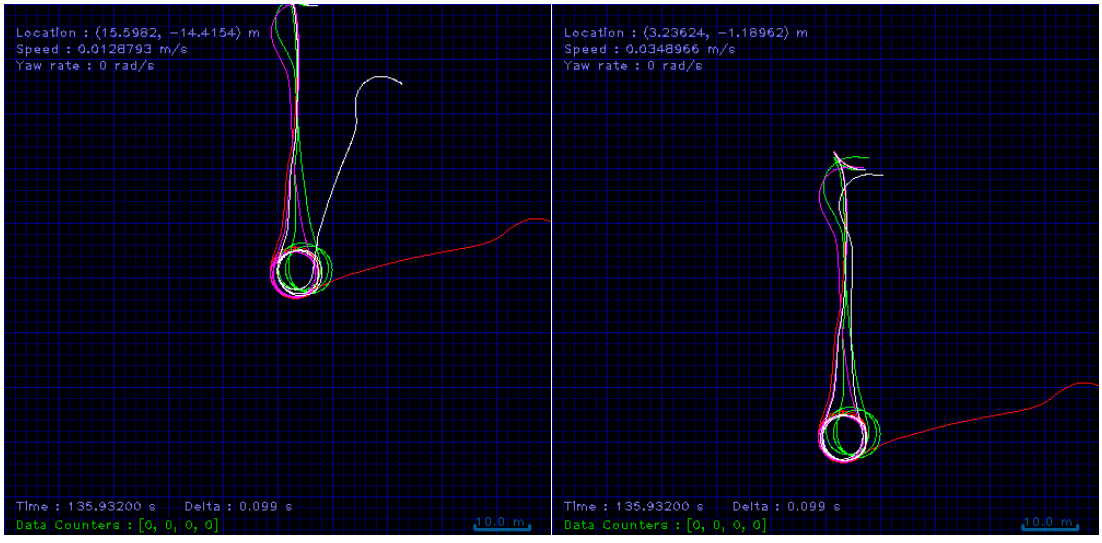


Figure 14 Comparison of EKF veh/Visual SLAM fusion results. The fused trajectory is shown in white, the other trajectories are: red - veh, green - vis, pink – Visual SLAM. (a) fusion with initial parameters, (b) with optimized parameter

4.5 Framework Integration

4.5.1 Framework description

The framework consists of components which produce data, consume data, or do both integrated in the ROS framework. The main producer is the Carla simulator which generates sensory data of various modalities. Carla-ROS bridge provides a bi-direction communication between Carla and the ROS ecosystem. In this ecosystem various algorithms have been implemented in the form of ROS nodes. These nodes subscribe to specific ROS topics, consume, and process the data, and finally generate an output. This output may be for example an estimated path or set of object detections. ROS framework is responsible for managing the communications and the lifecycle of the nodes. Finally, a REST API has been built for exposing the resources and providing remote access.

4.5.2 REST API

The goal of the REST API is to provide remote access to all the relevant resources of the framework. It is built using a Python Flask server, so it is meant to be deployed for development purposes. However, the switch to a production release does not requires much effort.

The following part describes the most important methods of the REST API. For each method the url is constructed by Flask's url plus the method's unique identifier. The Flask server's url is omitted in the examples.

4.5.2.1 Get available applications

Returns a list with all the available applications that can be executed remotely.

Method: GET /apps

Response: It returns a JSON array with the names (ids) of each available application.

4.5.2.2 Start application

It starts an application.

Method: GET /apps/<app_name>/start

Parameters: If the application is ROS then as a parameter is given the name of the node.

Arg	Type	Example Value
node	string	ros_dso

Response: If the operation is successful, it returns a 200 OK message.

4.5.2.3 Stop application

It stops an application.

Method: GET /apps/<app_name>/stop

Response: It returns a JSON array with the names (ids) of each available function**.** If the <app_name> id is not valid, it returns a 404 not found error code.

4.5.2.4 Get application's functions

It returns a list of the available functions of a specific application.

Method: GET /apps/<app_name>/functions/

Response: It returns a JSON array with the names (ids) of each available function**.** If the <app_name> id is not valid, it returns a 404 not found error code.

4.5.2.5 Run application's function

It executes a function of an application

Method: POST /apps/<app_name>/<func_id>

Response: If the operation is successful, it returns a 200 OK message.

4.5.2.6 Execute a scenario in Carla

It is special case of a function's application. The user provides a custom scenario in OpenSCENARIO format which will be used for the execution of the simulation.

Method: POST /apps/Carla/play_scenario

Parameters: The .xosc file which describes the scenario to be implemented in Carla and the record_filename argument which defines whether the simulation scenario will be recorded for future use or not.

Arg	Type	Example Value
record_filename	string	"True"
file	file	Scenario.xosc

Response: If the operation is successful, it returns a 200 OK message.

4.6 Metrics

Cooperative Localization: At time instant t actual position of vehicle i is equal to x_i and estimated $\hat{x}_i$. Self-location error (LE) of i is equal to $LE_{i\leftarrow i} = \| x_i - \hat{x}_i \|$. Location awareness error achieved by i is equal to $LAE_i = \left(\frac{1}{|N_i|}\right)\left(LAE_{j\leftarrow i}\right)^2$, for all vehicles j belonging to the neighborhood of i. Note $LAE_{j\leftarrow i}$ is the location error of j as measured by i using cooperative awareness solutions. The overall evaluation over simulation horizon T Is equal to $O - LAE = \left(\frac{1}{\mathrm{T}}\right) \sum_{t=1}^{T}\left(LAE_i^{(t)}\right)^2$.

Odometers:

1. Absolute pose error (APE) is based on the absolute relative pose between two poses $P_{ref,t}$, $P_{est,t} \in SE(3)$ at timestamp t : $E_t = P_{ref,t}^{-1} P_{est,t}$. Then, the translational and rotational APE will be equal to: $APE_t = \| trans(E_t) \|$, where $trans(E_t)$ is the translational part of E_t and$APE_t = \| rot(E_t) - I_{3\times3} \|_F$, where $rot(E_t)$ is the rotational part of E_t. Root mean square error (RMSE) is used to calculate APE over all time stamps: $RMSE = \sqrt{\left(\frac{1}{T}\right)\sum_{t=1}^{T} APE_t^2}$.
2. Relative pose error (RPE) compares the relative poses along the estimated and the reference trajectory. This is based on the delta pose difference: $E_{t,t+1} = \left(P_{ref,t}^{-1} P_{ref,t+1}\right)^{-1}\left(P_{est,t}^{-1} P_{est,t+1}\right)^{-1} \in SE(3)$. Translational, rotational and RMSE RPEs are calculated in the same way as APEs did.

2D Image-based scene analysis and understanding

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.

3D point cloud-based scene analysis and understanding

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[7]. 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

$$\text{AP}|_R = \frac{1}{|R|}\sum_{r\in R} \rho_{interp}(r)$$

averaging the precision values provided by $\rho_{interp}(r)$, according to [21]. 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

$$\rho_{interp}(r) = \max_{r':r'\geq r} \rho(r')$$

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.

4.7 Demo

4.7.1 Setup

The setup of the Carla ROS framework for the demo is the one described in the previous chapter. The resources are published via a REST API implemented by the Flask Server and the orchestration of the whole procedure is conducted by a Jenkins script.

4.7.2 Workflow

The process of utilizing the integrated simulation framework and running diverse automotive scenarios has been split into two parts which can be executed both serially and independently.

The first part deals with the data generation and their encapsulation into rosbag format. Here the user creates or selects a specific scenario that runs in Carla. The data produced by the spawned sensors are collected and stored. The second part describes the evaluation of several integrated algorithms against specific metrics. These algorithms are described in the previous chapters and most of them are integrated in the Carla-ROS ecosystem.

4.7.2.1 Creation of the rosbag

The goal of this process is the creation of a rosbag. Rosbags are file formats for storing data in ROS framework. They are created by subscribing to a custom set of topics and storing the received data messages. The work flow of the procedure is depicted in the following figure (Figure 15).

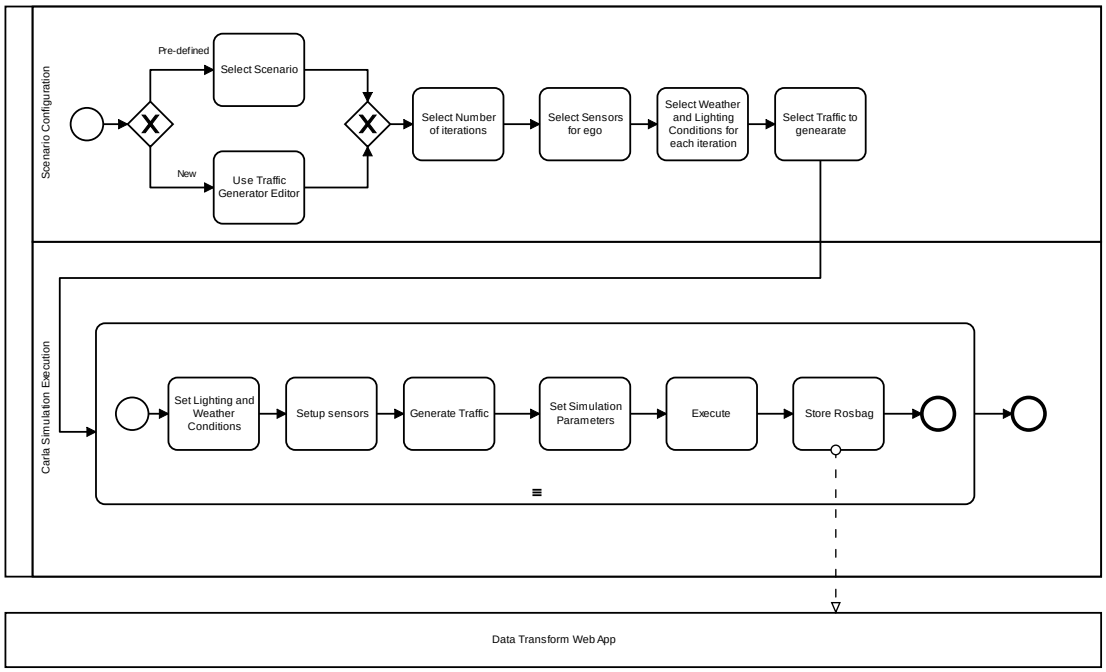


Figure 15 Rosbag creation

Initially, the user decides whether to create a new scenario or to select an existing one. The scenarios comply with the OpenSCENARIO format. The ASAM OpenSCENARIO [21] file format is used to describe the dynamic content of driving and traffic simulators. The primary use-case of OpenSCENARIO is to describe complex, synchronized maneuvers that involve multiple entities like vehicles, pedestrians, and other traffic participants. The description of a maneuver may be based on driver actions (e.g., performing a lane change) or on trajectories (e.g., derived from a recorded driving maneuver). Other content, such as the description of the ego vehicle, driver appearance, pedestrians, traffic, and environment conditions, is included in the standard as well.

```
<?xml version="1.0" ?>
<OpenSCENARIO>
    <FileHeader revMajor="1" revMinor="0" date="2021-12-28T00:42:28" description="Generated OpenSCENARIO File" author="QGIS OSCGenerator Plugin"/>
    <ParameterDeclarations/>
    <CatalogLocations/>
    <RoadNetwork>
        <LogicFile filepath="Town10HD"/>
        <SceneGraphFile filepath=""/>
    </RoadNetwork>
    <Entities>
        <ScenarioObject name="ego_vehicle">
            <Vehicle name="vehicle.audi.a2" vehicleCategory="car">
                <ParameterDeclarations/>
                <Performance maxSpeed="69.444" maxAcceleration="200" maxDeceleration="10.0"/>
                <BoundingBox>
                    <Center x="1.5" y="0.0" z="0.9"/>
                    <Dimensions width="2.1" length="4.5" height="1.8"/>
                </BoundingBox>
                <Axles>
                    <FrontAxle maxSteering="0.5" wheelDiameter="0.6" trackWidth="1.8" positionX="3.1" positionZ="0.3"/>
                    <RearAxle maxSteering="0.0" wheelDiameter="0.6" trackWidth="1.8" positionX="0.0" positionZ="0.3"/>
                </Axles>
                <Properties>
                    <Property name="type" value="ego_vehicle"/>
                </Properties>
            </Vehicle>
        </ScenarioObject>
    </Entities>
    <Storyboard>
        <Init>
            <Actions>
                <GlobalAction>
                    <EnvironmentAction>
                        <Environment name="Environment1">
                            <TimeOfDay animation="true" dateTime="2020-10-23T03:00:00"/>
                            <Weather cloudState="rainy">
                                <Sun intensity="0.0" azimuth="0.0" elevation="1.31"/>
                                <Fog visualRange="100000.0"/>
                                <Precipitation precipitationType="snow" intensity="0.9"/>
                            </Weather>
                            <RoadCondition frictionScaleFactor="1.0"/>
```

Figure 16 Snippet of OpenSCENARIO file

The traffic generator editor [22] is tool used for the creation of scenarios based on the OpenSCENARIO standard. It is based on the QGIS which is a free open-source geographic information system.

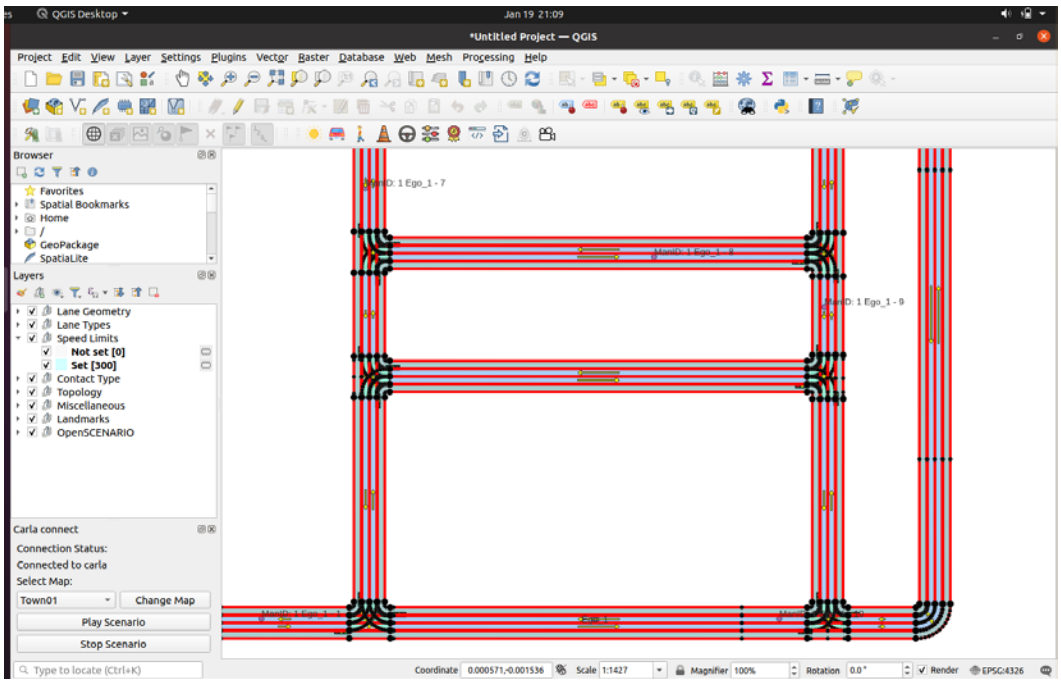


Figure 17 Scenario generator GUI

The basic steps for creating a new custom scenario are the following.

- Adding environmental variables
- Add vehicles
- Add pedestrians
- Add obstacles
- Add maneuvers
- Add KPIS
- Export to OpenSCENARIO format

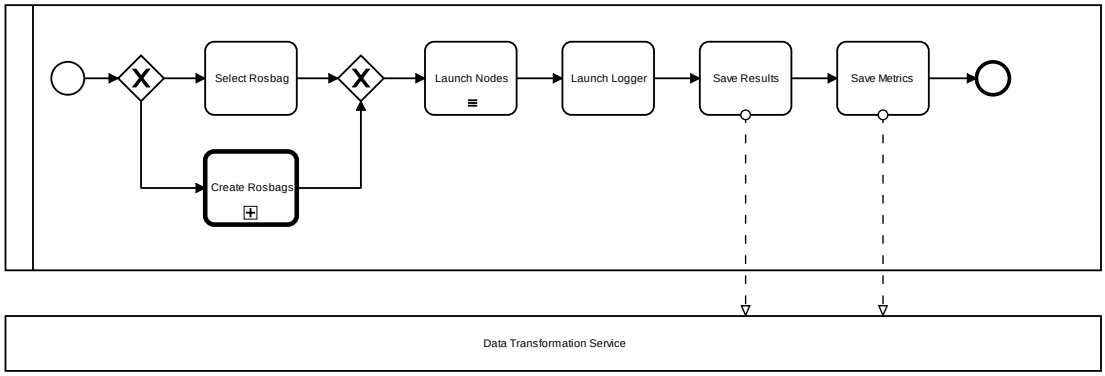
So, the user can generate a scenario which includes information like the map Carla will load, the path of ego vehicle, the attached sensors to the vehicle, additional vehicle and pedestrians, simulation duration, and specific events like a pedestrian crossing the road after the ego vehicle passes a specific point.

The simulation can be recorded and can be rerun with different weather and lighting conditions. Therefore, in the first instance the ego vehicle and all the other agents will follow specific trajectories which will be exactly reproduced in the latter executions.

Finally, the results will be recorded in a rosbag (one per simulation run) and then stored using the data storage and transformation service.

4.7.2.2 Algorithmic Evaluation

The second part of the includes the evaluation of the integrated algorithms by using specific metrics like the ATE and the RPE. The user either selects and already created rosbag or creates a new one. In the latter case, the procedure of the previous chapter is followed. Afterwards, the nodes that implement the algorithms under evaluation are launched. These, nodes consume data published under specific topics that were encapsulated in the rosbag and their results are stored using the data transformation service.



5 Security Runtime Monitoring and Management (SRMM)

5.1 Presentation of the Security Runtime Monitoring and Management

Security Runtime Monitoring and Management (SRMM) is the main security component of CPSoSAware, developed by the Task 4.3 - *CPSoSAware Security Runtime Monitoring and Management (SRMM) Design and Development*. Its objective is to detect attacks and anomalous behaviours that are a threat to the system. The module is based on the XL-SIEM asset that ATOS brought to the project. XL-SIEM is a cross-correlation Security Information and Event Management (SIEM) system, with complex multi-level security run-time monitoring.

The SRMM detects threats by correlating security events collected from different parts of the system. To gather the events, this component deploys monitoring sensors throughout the monitored infrastructure. At the same time, there are several correlation engines in the different system layers that evaluate whether the event sequences match configured rules, which model the behaviour that we want to detect. When a sequence of security events matches a rule, the associated alarm is raised. In addition, the alarm can be used as security events in the same SIEM or in another SIEM, performing a cross-correlation process. Finally, the system may execute actions that are associated with the specific alarm. Each SIEM deployed in the system may have its own rules, depending on the systems that it has to monitoring and the layer.

5.2 Position in the architecture and the interfaces

The SRMM is a hierarchy of XL-SIEMs where the lower SIEMs correlate local events, while the upper SIEMs have a more general view of the system. The first SIEMs receive the security events only from agents. On the other hand, the upper SIEMs can receive events from agents and lower SIEMs.

Agents are small services that monitor logs, parsing the raw information produced by the different sensors and normalising this information into the defined JSON format. For each source of information, there is a plugin that instructs the agent on which events are relevant and have to be parsed, how they have to be mapped into security events and which parts of the information have to be obfuscated. Finally, the agent sends the security events that it generates to the XL-SIEMs through TCP port 41000.

Regarding the architecture of the XL-SIEMs, there are three levels for this project:

- Lightweight SRMM: These SIEMs are in the lowest position in the hierarchy, deployed inside the vehicles. They receive the security events from sensors that monitor the different devices inside automobiles. These SIEMs have only a small set of rules that perform the local intelligence, thus they do not need a lot of computing capacity. This allows the system to maintain service to the vehicle even if the vehicle loses communication and becomes isolated. They also send the relevant security events to the upper XL-SIEMs, performing the first filter.
- Area SRMM: These XL-SIEMs manage the information of a physical area, receiving security events from the sensors deployed in it (the Lightweight SRMMs inside) and from other nearby area SRMMs. These security events are analysed, raising area alerts that are send to the Global SRMM and broadcasted to the Lightweight SRMM in the area. In addition, the relevant security events from below are forwarded to the Global SRMM, filtering out the useless information. These SIEMs broadcast the alarms generated by the Global SRMM to the Lightweight SRMMs below them.

- Global SRMM: It is a single XL-SIEM that sits at the top position and has a global overview of the CPSoSAware system. It receives security events from the Area SRMMs and the global agents. While its rules implement general intelligence, broadcasting alarms to all Area SRMM.

This hierarchy architecture enables distributed intelligence with high availability and workload sharing. Because the lower SRMMs process and filter events, improving performance, while they can work in isolation from higher SRMMs. More details of the architecture of SRMM can be found in D4.8 [23].

The nodes of the system use MQTT, which is a bidirectional protocol with two roles: publishers, which generate the information, and subscribers, which consume the information. The bidirectionality of the communication allows that the upper nodes to broadcast messages to the lower nodes. In addition, there is a broker that manages the communication, routing messages between clients. In the CPSoSAware project, the Lightweight SRMMs are the publishers for their corresponding Area SRMM. While the Area SRMMs are the publisher for the Global SRMM. In addition, each Area SRMM is the subscriber to nearby Area SRMMs. This communication mechanism allows Lightweight SRMMs, which are inside the vehicles, to switch between Area SRMMs quickly. This also allows a new Area SRMM to be deployed with a small reconfiguration of the neighbouring SRMMs. More details on the architecture and the communication system can be found in D4.7 [24].

5.3 Technologies and hardware requirements the SRMM

SRMM is a complex system, composed of several heterogeneous subsystems, more details the reader is referenced to D4.8 [23]. At the same time, each subsystem uses different technologies. To enable easier deployment and integration, all components of each SRMM is implemented within a docker container.. This solution also avoids conflicts between component dependencies. In the table below there is a list of SRMM components with the required versions and the necessary hardware requirements:

Component	Subcomponent	Software	Hardware
Global SRMM Area SRMM	Topology: • Zookeeper • Nimbus • Supervisors	• zookeeper:3.6.2 • storm:2.2.0 • storm:2.2.0	• Linux OS with Docker • 4 vCPU (8+ vCPU would be recommendable). • 4GB of RAM (8GB+ RAM would be recommendable) • 5GB+ of HDD/SDD
Lightweight SRMM	Light topology: • Zookeeper • Nimbus • Supervisors	• zookeeper:3.6.2 • storm:2.2.0 • storm:2.2.0	Hardware requirements are lower than other SRMMs, however load testing is pending.
SRMM BBDD		mariadb:10.2	• Linux OS with Docker • 1 vCPU (2+ vCPU would be recommendable) • 256MB of RAM (512MB + RAM would be recommendable) • 5GB+ of HDD/SDD

SRMM Dashboard		php:5.6.38-apache	• Linux OS with Docker • 1 vCPU (2+ vCPU would be recommendable). • 256MB of RAM (512MB + RAM would be recommendable) • 5GB+ of HDD/SDD
Agent		debian:stretch-20201209 python 2.7	• Linux OS with Docker • 1 vCPU (2+ vCPU would be recommendable). • 128MB of RAM (256MB + RAM would be recommendable) • 5GB+ of HDD/SDD
MQTT broker		rabbitmq:3.7.14-management	• Linux OS with Docker • 1 vCPU (2+ vCPU would be recommendable). • 128MB of RAM (256MB + RAM would be recommendable)

5.4 Technical details about the interfaces

The SRMM is a security component that is distributed through whole system and receives information from many sources, so it has several input interfaces throughout the system. However, due to the heterogeneity of the devices in a CPSoS, it is necessary that all sensors generate the output in a common format. For this reason, the SRMM is complemented by an agent, which transforms the raw information generated by the different sensors to a standard security event. Figure 18 depicts the fields of these events. The XL-SIEMs receive them on TCP port 41000. Deliverable D3.5 [25] describes how the agent maps the raw information into security events.

```
"a": {“type”: <string>,
"date": <string>,
"device": <string>,
"interface": <string>,
"plugin_id": <integer>,
"plugin_sid": <integer>,
"src_ip": <string>,
"dst_ip": <string>,
"src_port": <string>,
"dst_port": <string>,
"userdata1": <string>,
"userdata2": <string>,
"userdata3": <string>,
"userdata4": <string>,
"userdata5": <string>,
"userdata6": <string>,
"userdata7": <string>,
"userdata8": <string>,
"userdata9": <string>,
"log": <string>,
"fdate": <string>,
"tzone": <string>,
"event_id": <string>,
"username": <string>,
“password”: <string>,
"filename": <string>,
"organization": <string>
}
```

Figure 18. XL-SIEM event data: JSON format

On the other hand, the SRMM produces alarms as output, Figure 19 depicts JSON format of the alarms. They may be displayed in the dashboards of the different level, warning of a threat; they can be sent through a MQTT channel; or the action associated with the alarm can use any API to communicate an alarm. This last is method used to communicate information to the CARLA Simulator and V2X Simulator, as described in D1.4 [2].

```
{"AlarmEvent": {
        "DST_IP_HOSTNAME": <string>,
        "RELATED_EVENTS": <string>,
        "DST_IP": <string>,
        "PLUGIN_NAME": <string>,
        "SRC_IP": <string>,
        "PRIORITY": <integer>,
        "RELIABILITY": <integer>,
        "SUBCATEGORY": <string>,
        "USERDATA3": <string>,
        "USERDATA4": <string>,
        "PLUGIN_SID": <string>,
        "USERDATA1": <string>,
        "USERDATA2": <string>,
        "ORGANIZATION": <string>,
        "CATEGORY": <string>,
        "PLUGIN_ID": <string>,
        "USERNAME": <string>,


"FILENAME": <string>,
"BACKLOG_ID": <string>,
"RELATED_EVENTS_INFO": {List of <Event>},
"PROTOCOL": <integer>,
"RISK": <integer>,
"SRC_PORT": <integer>,
"SENSOR": <string>,
"SRC_IP_HOSTNAME": <string>,
"SID_NAME": <string>,
"USERDATA7": <string>,
"DATE": <string>, YYYY-mm-dd HH:MM:SS
"USERDATA8": <string>,
"USERDATA5": <string>,
"USERDATA6": <string>,
"PASSWORD": <string>,
"USERDATA9": <string>,
"DST_PORT": <integer>,
"EVENT_ID": <string>}}
```

Figure 19. XL-SIEM alarms JSON data format

5.5 Application on the Automotive use case

In deliverable D4.8, there are four demonstrations of the SRMM capacities. The first is a detection of a Denial of Service (DOS) where an attacker infects a vehicle that will start to send a high number of messages to the channel. This unusual traffic causes an overload of communication capacities, which results in other vehicles not be able to access the services. When this happens, the SRMM system detects the situation and raises an alarm.

The other three demonstrations are examples of how the SRMM can detect and mitigate an attack against a device inside the vehicles. In the second case, a sensor detects a firmware update produced by an attacker. The Lightweight SRMM compares the new version of the firmware with manufacturer's version list, and if it does not match, the XL-SIEM raise an illegitimate update alarm, which is the detection phase.

Associated with the alarm, an action is launched that updates the devices with the latest version of the manufacturer's firmware, mitigating the attack.

The third case is an extension of the previous demonstration in which several vehicles in an area are infected in the same way. The attacks are mitigated by the Lightweight SRMM, raising the illegitimate update alarm. When multiple alarms arrive at the Area SRMM, an alarm is raised that has a mitigation action associated with it. The action blocks the source (URL and IP) of the malicious firmware on all vehicles within the area.

Finally, the last demonstration mitigates and solves possible future attacks. In this case, the illegitimate update alarms come from several areas, so the Area SRMM cannot mitigate the attack because each XL-SIEM does not receive enough security events to trigger an alarm. All these security events reach the Global SRMM which raises an illegitimate update alarm that has associated blocking the malware source in all vehicles that belong to the system, as in the previous case. In addition, this SRMM create a report with the details of the attack, which should be notified to the manufactures to correct the vulnerability. When a manufacturer releases a new version of the firmware, the Global SRMM broadcasts an alarm with the information of the update. At that point, each Lightweight SRMMs have to evaluate whether its vehicle has the affected device and launch the update if necessary.

5.6 Implementation and future work

ATOS brough to the project the XL-SIEM, which is the correlation engine of events, and the agent to normalise the raw events from the sensors. With these two components, ATOS also bought a set of plugins for common sensors and the rules that detect threats using the events from these sensors. In addition, ATOS has integrated RabbitMQ software, which implements the MQTT protocol used in the project, into the communications output.

During the project, ATOS has developed plugins for the sensor created by other partners and the rules to handle the security events generated by these sensors. Until the end of the project, ATOS is going to connect the Lightweight SRMM using MQTT brokers that are deployed inside vehicles by I2CAT, more details in D4.7 [24]; develop plugins and rules for the sensors that the partners release; and implement the communication with the CARLA and V2X simulators.

6 Hardware Acceleration Components

In this section, the implementation and integration aspects of technical components related to hardware acceleration are presented. A brief description of these components, as reported in more details in D1.4 [2] , is given below.

Pocl-remote (TC2.2.2): Scalable distributed OpenCL runtime layer with P2P event synchronisation capabilities.

ML Hardware Accelerator IP Cores (TC2.3.1): FPGA-based IP core components (interfaces) focused on ML/DNN computations. The FPGA IP cores will be automatically generated from ONNX based ML/DNN models by using an appropriate ML framework. The IP cores will be seamlessly integrated in the PoCL-based OpenCL run-time system.

Modelling Orchestration Tool (TC2.5.1): The modelling orchestration tool captures the CPS overall, manages individual CPS inputs and outputs between other CPSs, and orchestrates the CPSoS components in order to achieve a model of models.

User Behaviour Monitoring (TC3.1.3): The user behavioural monitoring will be based on CPSoSaware's collaborative sensory multi-modal fusion mechanism and will be based on algorithms for physiological and behavioural monitoring that will facilitate the evaluation of cognitive load/situational awareness development of a smart sensing module to allow inertial and optical sensor fusion, providing 6DoF pose estimation, thus dealing with occlusions and drifts. The specificities of the algorithms will be defined by the system requirements and use cases.

AI Acceleration (TC3.1.4): DCNNs achieve ground-breaking performance in a great variety of applications, including classification tasks such as object recognition. However, DCNNs are computationally expensive, meaning that they usually demand high-performance platforms for their implementation. The goal is the study of DCNN acceleration / compression techniques for their effective implementation in embedded platforms, lower the computational cost (number of operations, storage requirements) with the least possible loss in accuracy. Specifically, our efforts are focused on pruning and sharing techniques that can achieve considerable acceleration without significant performance loss and can be applied to pre-trained DCNNs. These techniques are orthogonal and could potentially be combined.

Pocl-accel (TC3.2.1): This is a Generic OpenCL driver (for POCL) to interface with custom devices (hardware accelerators) from the OpenCL API.

TCE (openasip.org) Soft Cores (TC3.6.1): Customised processors designed using TTA-Based Co-design Environment (TCE), an open source application-specific instruction set toolset based on the transport-triggered architecture (TTA). Various hardening features can be added via replication of functionality and special instructions.

OpenCL Wrapper for Hardware IP Cores (TC4.1.1): OpenCL kernel description interface to associate Hardware IP cores with the OpenCL models.

Profiling (TC4.1.2): Profiling for a highly heterogeneous platform consisting of multicore ARM processor, ASIP processors as well as FPGA fixed logic IP. FPGA logic is a "morphable" computation resource without predefined computational capabilities. All SW nodes will be handled by PoCL remotely enabling dynamic remapping and re-scheduling opportunities.

Optimization (TC4.1.3): This component aims to provide all necessary optimizations in order to reconfiure and redesign the System's CPSs/CPHSs so as to holistically match the systemic design and operational goals/parameters achieving reliability, robustness, responsiveness, CPS/CPHS criticality, energy efficiency, and security/trust.

Commissioning of Hardware Components in CPSs: The Developed Hardware components after HW/SW partitioning will need to be deployed in the CPS. We focus on the dedicated HW accelerator components designed in other tasks and we aim at structuring the deployment/commissioning mechanism in the CPS SoC FPGA Fabric. In T4.6 we will focus on the commissioning mechanism from the System layer perspective while in task T5.2 we will focus on the commissioning mechanism infrastructure (support) at the CPS layer (in each CPS).

These components have been applied and evaluated on two use cases developed by Up: a)a CNN application implementing from a single Handwritten Digit Recognition (HDR) to a complicated SqueezeNet classification and b) Driver Status Monitoring (DSM) system that detects driver drowsiness by counting the yawnings and sleepy eye blinks of the driver. The integration to higher level applications of both the use cases as modules is described by the respective input/outputs. Then, for each component (as listed in D1.4 [2]) that has been implemented in the two use cases, its input/output data structures are also referenced in order to comprehend how this component has been integrated within a use case and how these components are communicating with each other.

6.1 CNN module implementing HDR, SqueezeNet

Any application that concerns the commissioning of CPS components that operate in parallel is appropriate for implementation on the PoCL framework (OpenCL systems implemented on FPGAs, GPUs, etc).

The first use case examined by UoP concerns the implementation of Deep NNs (DNNs) on the PoCL framework. More specifically, Convolutional NNs (CNNs) have been implemented in the PoCL environment with extensions e.g., the DMA support of PoCL for the fast input argument passing. The simpler CNN that was implemented concerned the recognition of handwritten characters (trained on the MNIST dataset). It consisted of 2 fully connected layers. A more complicated CNN was based on SqueezeNet v1.1 for the classification of 1000 object categories using the ImageNet 2012 dataset. The architecture of this CNN consisted of 18 convolution, 3 max-pool, and 1 average pool layers.

Input: Grayscale image with 28×28 (HDR) or RGB image with 227×227 (SqueezeNet) resolution.

Output: a vector of 10 (HDR) or 1000 (SqueezeNet) values representing the confidence in each category. These values are implemented either as 32-bit floating points or as int8.

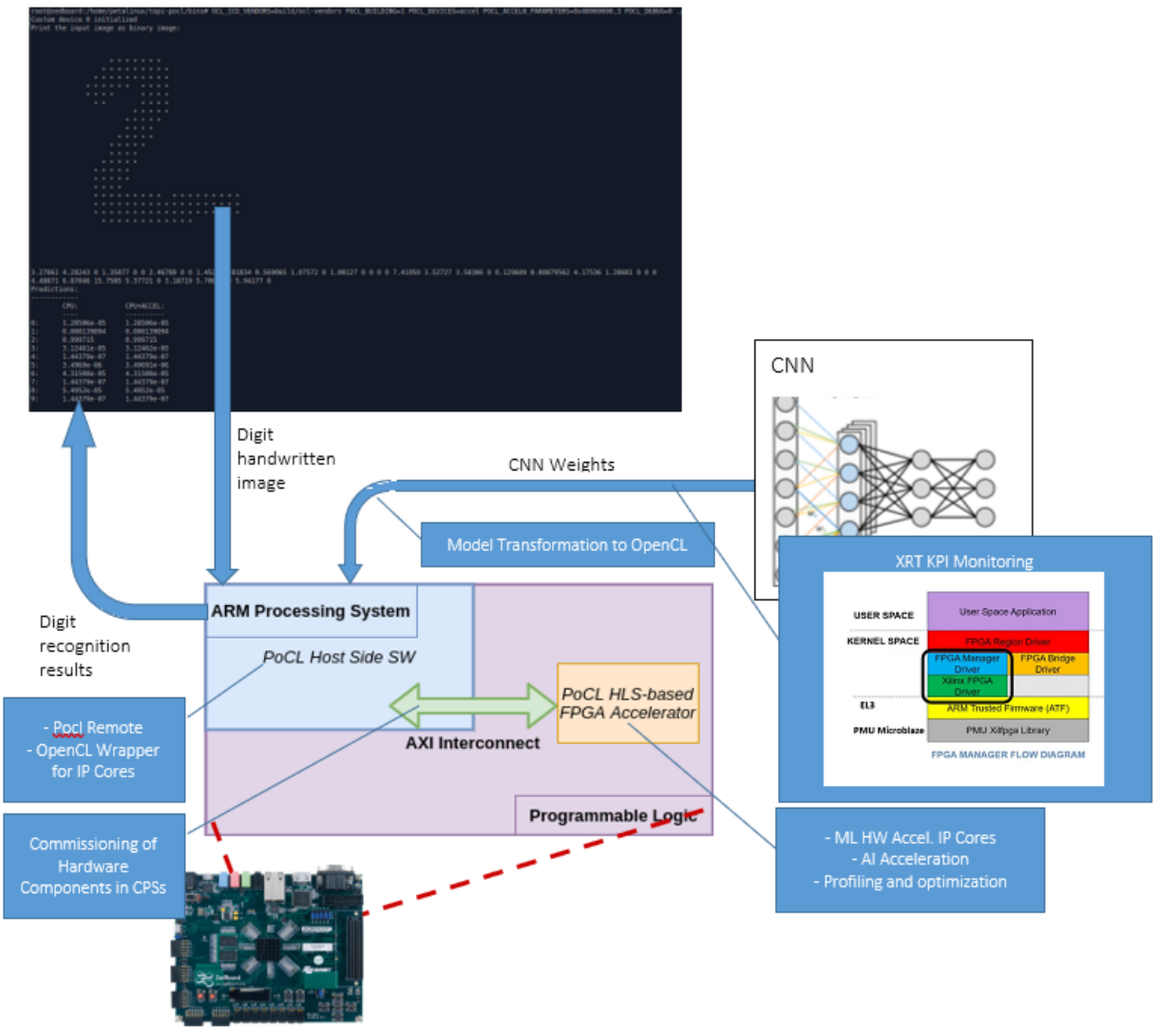


Figure 20 HDR application with Pocl

The environment of the HDR application is shown in Figure 20 with the TC annotated in blue boxes to make clear on how they are related to the various application modules.

The following TC components listed in D1.3 have been integrated in the CNN PoCL framework:

Pocl-remote: PoCL has been used to remotely invoke the CNN core that performs handwritten character recognition

Input: Grayscale image with 28×28 (HDR) or RGB image with 227×227 (SqueezeNet) resolution and a command to start classification.

Output: a vector of 10 (HDR) or 1000 (SqueezeNet) values representing the confidence in each category. These values are implemented either as 32-bit floating points or as int8.

ML Hardware Accelerator IP Cores: FPGA-based IP core has been used to implement CNN/DNN computations that are seamlessly integrated in the PoCL-based OpenCL run-time system.

Input: Grayscale image with 28×28 (HDR) or RGB image with 227×227 (SqueezeNet) pixel values and matrices with the CNN weights. The HDR NN is implemented as 2 layers, the weights of the 1st layer are 764×30, and of the 2nd layer 30×10. In the SqueezeNet 1.24 million weights have to be provided. Weights represented either as 32-bit floating points or int8.

Output: a vector of 10 (HDR) or 1000 (Squeezenet) values representing the confidence in each category. These values are implemented either as 32-bit floating points or as int8.

Model Transformation to OpenCL: The NN model representation was adapted to be compatible with OpenCL

Input: The HDR NN is implemented as 2 layers, the weights of the 1st layer are 764×30, and of the 2nd layer 30×10. Number of weight values in the SqueezeNet CNN: 1.24 million.

Output: OpenCL/C/C++ code that implements this CNN architecture on FPGA.

Xilinx XRT KPI Monitoring: XRT facilities that support dynamic reconfiguration have been employed.

Input: Xilinx xclbin files that implement FPGA core functions.

Output: Indication about the current xclbin file that has been used.

AI Acceleration (implementation of the "ML Hardware Accelerator IP Cores" component as AI-CNN): DNN/CNN operations are appropriate for hardware acceleration since there aren't strict requirements for data transfer while the computational overhead is high. These NNs usually demand high-performance platforms for their implementation with the least possible loss in accuracy. Pre-trained NNs have been used in our applications.

Input: Grayscale image with 28×28 (HDR) or RGB image with 227×227 (SqueezeNet) resolution.

Output: a vector of 10 (HDR) or 1000 (SqueezeNet) values representing the confidence in each category. These values are implemented either as 32-bit floating points or as int8.

Pocl-accel : PoCL based HW accelerator has been implemented for the acceleration of CNNs on FPGAs. This module actually calls the AI Acceleration component passing as arguments the image and the CNN weights. The outcomes of the AI Acceleration component are feeding back the calling Pocl-Accel component.

Input: Grayscale image with 28×28 (HDR) or RGB image with 227×227 (SqueezeNet) resolution

Output: a vector of 10 (HDR) or 1000 (SqueezeNet) values representing the confidence in each category. These values are implemented either as 32-bit floating points or as int8.

OpenCL Wrapper for Hardware IP Cores : OpenCL kernel description interface is used to associate the Hardware IP cores that implement the computationally intensive operations of a CNN.

Input: Grayscale image with 28×28 (HDR) or RGB image with 227×227 (SqueezeNet) resolution. Argument buffers implemented in C/C++ format.

Output: a vector of 10 (HDR) or 1000 (Squeezenet) values representing the confidence in each category. These values are implemented either as 32-bit floating points or as int8. These arguments are described as OpenCL buffers/pointers.

Profiling & Optimization : The selection of the computationally intensive CNN operations to be implemented in HW was performed based on profiling at various levels (application level, HLS estimations, XRT real time profiling).

Input: C/PoCL code chunks.

Output: Utilisation, measured latency of the code chunks, required memory and power consumption, resources (if implemented in hardware).

Commissioning of Hardware Components in CPSs : An XRT based "system-call" method that allows dynamic reconfiguration of the hardware in the PoCL platform has been successfully employed as will be described in the next paragraphs.

Input: Xilinx bit files implementing hardware functions.

Output: Downloading of the bit files to FPGA.

6.2 DSM module

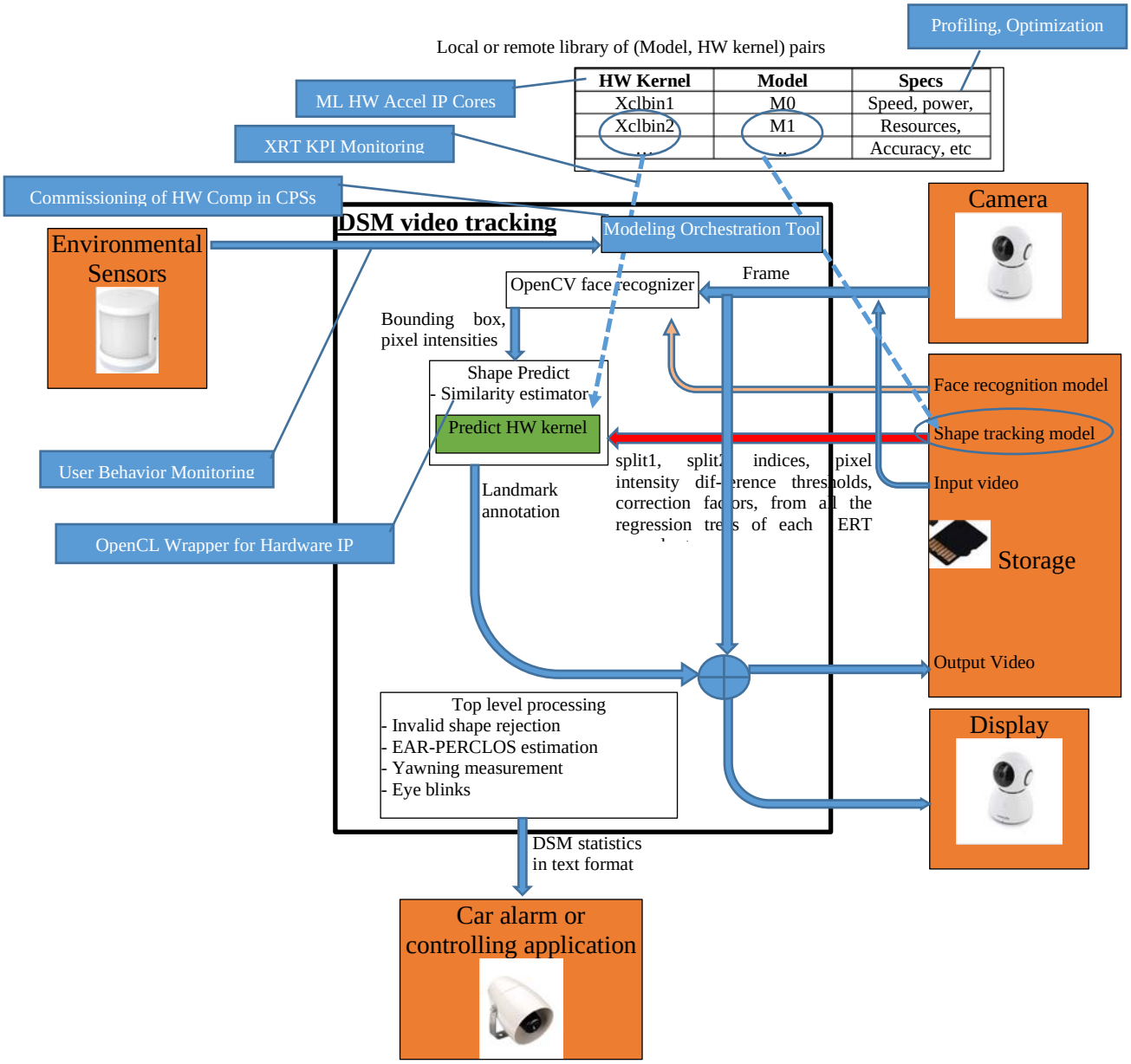


Figure 21 DSM Video for tracking application architecture

The second use case concerns a Driver Status Monitoring (DSM) application which is responsible for detecting driver drowsiness and distraction from yawning, eye blinks, and head movement. This application is based on an Ensemble of Regression Trees (ERT). The DSM application has been developed using Xilinx Vitis platform and Xilinx RunTime (XRT) library for a ZCU102 target board. The detailed architecture of

the DSM application is described in the deliverables D3.2, D4.6 and D5.1. The architecture of this DSM video tracking application with the technical components annotated in blue boxes is displayed in Figure 21.

Input: Video or camera source of any common format (mp4, avi, etc), OpenCV face detector (classifier xml format), pretrained tracker for shape alignment (bin format).

Output: video with the annotated landmarks and information about yawnings, eye blinks encountered stored in SD card of the FPGA, an ASCII text stream informing periodically about the facial shape landmark coordinates, the encountered yawningns, eye blinks, the rejected frames and the metrics such as PERCLOS, MAR that are used to recognize yawnings, and eye blinks.

The following TC components have been integrated in the DSM application:

ML Hardware Accelerator IP Cores: Computationally intensive ERT processing operations implemented as HW kernels. More specifically, the Regressor::Tracker() and the nested Tree::Tracker() routines of the DEST package were implemented in hardware.

Input: Current facial shape estimation (68×2 32-bit floating point numbers), tree node split information (2 buffers of Trees × Nodes uint32), tree node thresholds (Trees × Nodes 32-bit floating point numbers), sparse image pixel intensities (P bytes), correction factors (Tree×Nodes×68×2 32-bit floating point numbers).

Output: The updated current facial shape estimation (68×2 32-bit floating point numbers).

OpenCL Wrapper for Hardware IP Cores: OpenCL used to invoke a HW kernel from the top level SW

Input: A frame of a video or camera source of any common format (mp4, avi, etc), OpenCV face detector (classifier xml format), pretrained tracker for shape alignment (bin format).

Output: The inputs of the ML Hardware Accelerator IP Cores in OpenCL format i.e., current facial shape estimation (68×2 32-bit floating point numbers), tree node split information (2 buffers of Trees × Nodes uint32), tree node thresholds (Trees × Nodes 32-bit floating point numbers), sparse image pixel intensities (P bytes), correction factors (Tree×Nodes×68×2 32-bit floating point numbers).

Xilinx XRT KPI Monitoring: XRT is used to dynamically reconfigure the HW kernels

Input: Xilinx xclbin files that implement FPGA implemented core functions, command for the selection of the appropriate xclbin.

Output: Indication about the current xclbin file that has been used.

Modelling Orchestration Tool: Based on environmental conditions indicated by the system inputs, different ERT models implemented by the corresponding HW kernels may be configured.

Input: Sensor inputs (e.g., light sensor to detect night/day, or real time clock) or external Convolutional Neural Networks recognizing male of female driver

Output: command (integer code) to the Xilinx XRT KPI Monitoring module to select the appropriate hardware kernel (xclbin file in Xilinx FPGAs).

User Behaviour Monitoring: The target of the DSM module is actually to monitor the behaviour of the driver and more specifically if he/she is yawning, his eyes are blinking, he is experiencing a microsleep, or if he is distracted. This component can be used to provide the inputs of the Modelling Orchestration Tool.

Input: The input of the DSM module: video or camera source of any common format (mp4, avi, etc), OpenCV face detector (classifier xml format), pretrained tracker for shape alignment (bin format), sensors (see Modelling Orchestration Tool input).

Output: The inputs of the Modelling Orchestration Tool.

Profiling: The selection of DSM operations that were appropriate for HW implementation were selected through profiling at various levels (application level, HLS estimations, XRT real time profiling).

Input: ERT algorithm functions such as Estimate Similarity Transform, Landmark Position Prediction.

Output: Utilisation, measured latency of the functions, required memory and power consumption, resources (if implemented in hardware).

Optimization: The alternative pairs of (ERT models, HW kernels) are selected based either on optimization targets (see T4.1 for more details) or environmental conditions with dynamic reconfiguration.

Input: Library of (model, kernel) pairs: each model is a pretrained tracker (bin format in DEST package) and each kernel is in Xilinx xclbin format. The constraints and the optimization goal are also inputs to this module.

Output: The selected (model, kernel) pair that achieves the optimization goal.

Commissioning of Hardware Components in CPSs: Dynamic reconfiguration of HW kernels through XRT by switching in real time between bitstreams that implement different HW kernels.

Input: Xilinx xclbin files implementing hardware functions.

Output: Downloading of the xclbin files to FPGA.

More details about the integration of the DSM application can be found in the deliverables of T3.6 (dynamic reconfiguration), T4.1 (HW/SW partitioning optimization) and T5.1 (specifications of the generated HW kernels in the DSM application).

7 Intra – Communication Layer

The system intra communication layer (SICL) undertakes the responsibility to establish efficient and reliable wireless communication technologies between the CPSoS and the CPSs. It consists of two essential technical components, the intra – communication simulation tool and the intra – communication manager. The intra – communication tool lies on the CPSoS layer of the CPSoSaware architecture while the intra – communication manager on the CPS/CPHS layer. This chapter will present the interfaces of the components that allow integration with other components of the CPSoSaware system along with the activities performed towards this direction.

7.1 Intra – communication simulation component

The intra – communication simulation tool (TC2.2.1) is designed and implemented to match network requirements imposed by the application and deployed to CPSoS, to proposed network technologies and configurations (e.g., modulation, signal strength, duty cycle etc.) and network topologies. The tool is based on the NS3 simulator, and it aims to accelerate the experimentation of models of dominant wireless protocols for intra-communication, e.g., BLE, ZigBee/802.15.4, Wi-Fi.

The aim of this tool is to provide the functionality of the dominant NS3 network simulator as a service, and facilitate the iterative execution of simulations that aim on the near optimal configuration of the network interfaces in favour of the application requirements. This approach dictated the design and implementation of interfaces that support the interaction with the NS3 through respective input and output ports. The input ports of the implementation are used to trigger simulations and feed the simulator with the respective scenario and model configuration under evaluation. This interface follows the latest principles of integration patterns that apply in state-of-the-art web based and distributed systems. In that context a RESTful API was designed and implemented allowing the remote execution of REST calls that trigger NS-3 simulations with dynamic configuration feeds. The definition of this API is given in Figure 22 while Figure 23 describes the data structures to be transferred via this API REST calls. To facilitate this integration with the rest of the components, this API is published under a public URL available to the consortium partners:

http://ns3.simulations.manager.esdalab.ece.uop.gr/v1/webjars/swagger-ui/index.html?configUrl=/v1/v3/api-docs/swagger-config

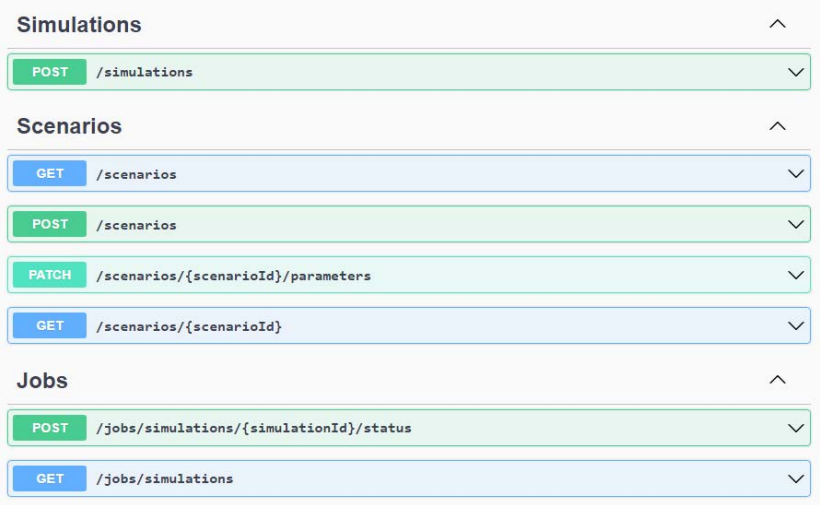


Figure 22 NS3 REST API

This interface allows external components to trigger the execution of simulation scenarios and get notified when the simulations are over through the respective callback. Apart from the potentials of integration with other simulators, this approach allowed integration with the Jenkins automation server which part of the TC4.6.1 component as described in Section 3 of this deliverable. In that sense, Jenkins is able to initiate simulations and get awareness about the status of the simulation execution (queued, running, completed).

A second interface was implemented for exposing the simulation traces of the NS3 simulator. Thus, 3rd party systems could extract these traces and utilize the behaviour/traffic of a network for a specific configuration of the wireless technology. Besides the generation of files with the traces of the simulations, stored in the filesystem, a Java based REST client consumes the storage and transformation engine's (SAT) REST API in order to store results of the simulations in the database. CPSoSaware or other external components are able to consume these traces/results either through the extracted files or the database. The details of this Simulation as a Service approach of the NS-3 are given in Deliverables 1.4 [2] and 4.2 [26].

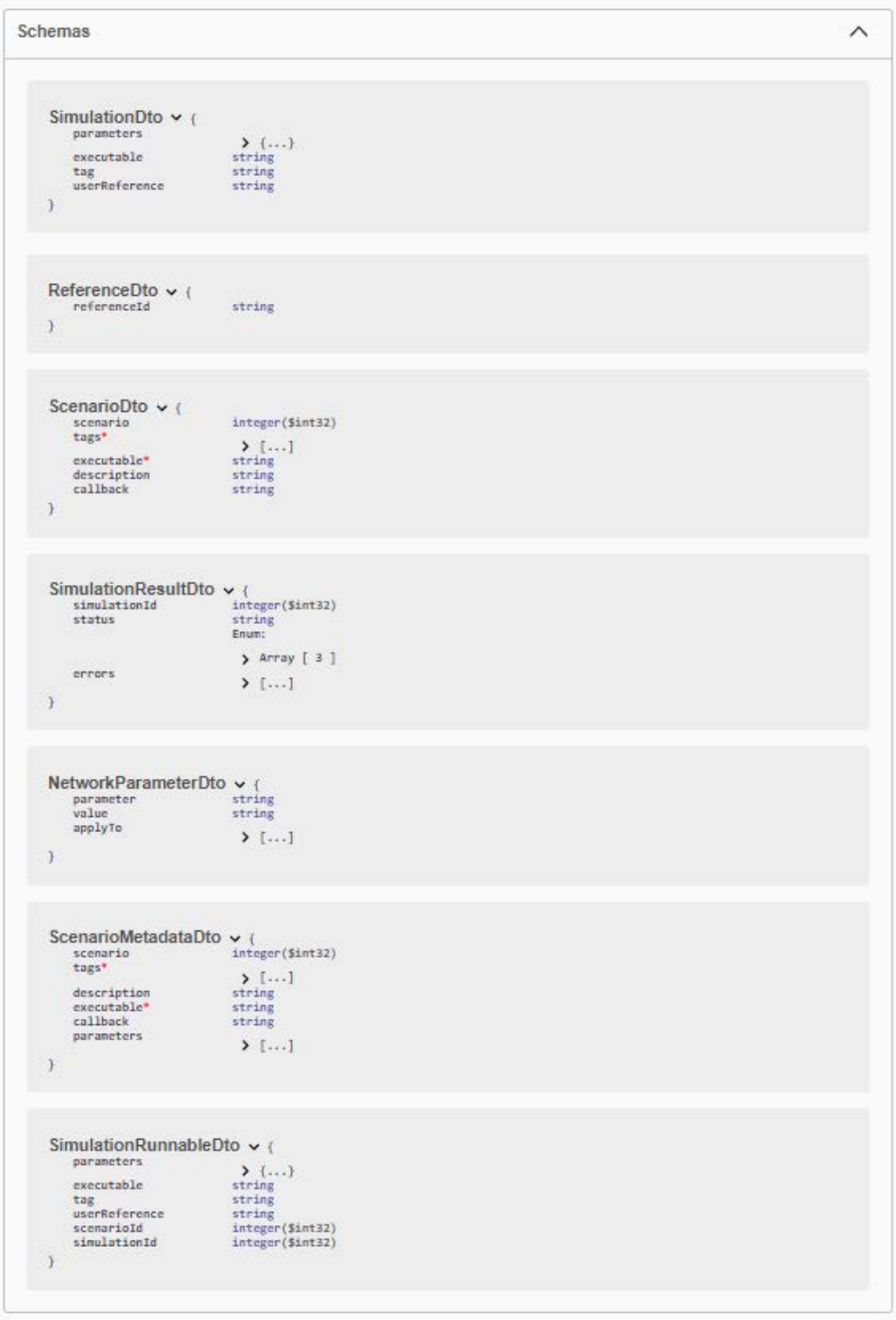


Figure 23 API Data Transfer Objects (DTOs)

7.2 Intra – communication manager

The intra – communication manager incorporates mechanisms to apply network configurations and supervise their performance in a real deployment along with handling and forwarding the network traffic. Particularly, the functionality of intra – communication manager as described in D3.2 [27] can be summarized in the following two pillars:

1. the deployment / commissioning: This component is responsible to deliver to the target system the configurations of intra – communication wireless interfaces.
2. the execution mechanism: This component is responsible to handle RX/TX of data over the available intra – communication wireless interfaces.

The first steps of integrating this intra – communication manager with other components was to define and implement the interface for transferring the various network configurations to the target platform.

The basic prerequisite for the target platform, for this 1st version of the configuration of network interfaces, was to support the execution of the Linux Operating System. This allowed to build an API on top of the wireless interface's drivers and kernel's network settings and interact with the available configuration options. The implementation of this API is using the Python programming language and take advantage of utilities such as *iwconfig* [28] and *nmcli* [29] and *sysctl* [30].

The interface is based on the MQTT message passing protocol. This approach allowed the asynchronous transmission of network configurations to more than one target platflorms. The network configuration data transfer objects (DTOs) are published to respective MQTT topics where the target platforms are subscribed to listen. This MQTT topics follow the formats shown in Table 1.

Table 1 Network configuration MQTT topics

/network/configuration/#	apply configuration to all subscribers
/network/configuration/<device_id>	apply configuration only for the specified device id
/network/configuration/<device_id>/<interface>	apply configuration only for specific network interface
/network/configuration/<device_id>/<interface>/<parameter>	apply value to specific parameter of network interface

In parallel, a monitoring mechanism is running on the device is responsible to capture periodically the network performance and report it back to the CPSoSaware system layer. The performance is described by metrics such as packet loss, throughput and transmission delays. These performance vectors can then be

used for the evaluation and optimization of the applied network configurations. The performance vectors are published to the respective MQTT topics as show in Table 2.

Table 2 Performance reporting MQTT topics

/network/performance/#	Listen to network performance vectors from all target platforms
/network/performance/<device_id>	Listen to network performance vectors from specific target platform

This approach allows the automation mechanisms presented in Section 3 to integrate with the target platform and facilitate the commissioning of network configurations as throughout the simulation phase described in Section 7.1.

The second pillar of handling and forwarding the network traffic through the wireless interfaces is a work in progress. It be finalized during the upcoming months and reported to the 2nd revision of this deliverable.

7.3 Demo

The described components will be demonstrated through standalone demonstrators for various patterns of traffic. The scope is that proposed solutions can handle heterogeneous traffic with regards to data volume and quality of service. This network will span from small volume of data, such as sensor reading to larger volumes that regard images, sound of video streams. In parallel, at least two different wireless interfaces will be supported (WIFI, BLE). The scope of the demonstrator will be to present that the intra – communication manager can handle efficiently all the generated traffic patterns by utilizing the available wireless network interfaces and through their optimization. This will be manifested through network performance statistics that will be captured and transmitted periodically.

8 AV Simulator

Robotec integrated two modules used in validation of Autonomous Driving algorithms:

- RoSi simulation platform – Unity based simulator used simulation of sensors and movement of all traffic agents
- V2X Simulator - a co-simulator used for modelling of communication between traffic agents

Thanks to integration of the mentioned simulators, it is possible to simulate and validate cooperative awareness algorithms (e.g. Cooperative Localization, Extended Perception). V2X Simulator creates a copy of the environment simulated in the AV Simulator (Figure 24), and thanks to ROS2 integration can be easily deployed on the other machine, what reduces the number of computations performed on the main simulation machine.

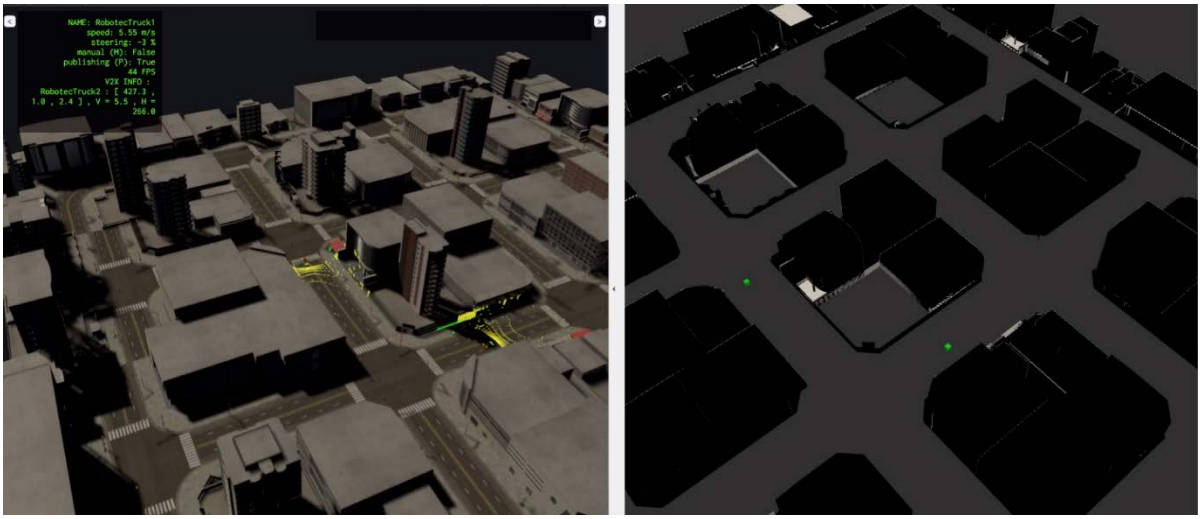


Figure 24 Visualization of Rosi Simulator (left), and V2X Simulator (right). The same situation is replicated in two simulators

8.1 Integration interface

Communication interface between RoSi and V2X simulator (Figure 25) is based on ROS2.

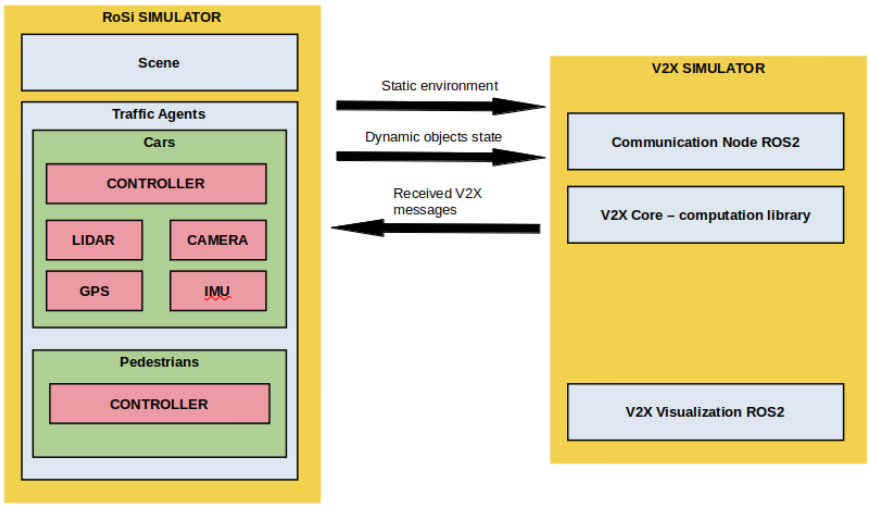


Figure 25 High level integration diagram of RoSi and V2X Simulator

For the communication of 2 simulators, the following custom ROS2 messages were created:

- ShapesArray.msg

  geometry_msgs/Polygon[] shapes

  Array of polygons are used to share the representation of the environment (all static objects) as well as meshes of dynamin objects (cars, trucks etc.)

- VehicleState.msg

  std_msgs/String name

  std_msgs/Float32 x

  std_msgs/Float32 y

  std_msgs/Float32 z

  std_msgs/Float32 velocity

  std_msgs/Float32 heading

VehicleState message is send from RoSi to V2X Simulator to replicate states of dynamic objects in the communication simulator, then to form actuall V2X messages and model the propagation of the messages. For Extended Perception scenarios, perception of the objects will also be shared using this ROS2 message

- V2Xmsg.msg

  std_msgs/String vehicle_name

  VehicleState[] vehicles_states

V2Xmsgs represents messages that are successfully received by an object in V2X Simulator and are sent back to the main AV simulator (RoSi) to enable using extended cooperative awareness by Autonomous Driving algorithms controlling the behaviour of traffic agents

9 Conclusions

Task 5.2 tackles the CPSoSaware integrated platform. This work in progress is presented in this 1st deliverable that reports a subset of the CPSoSaware components and their integration progress, with respect to the implemented interfaces and data structures that are exchanged among the integrated entities. These components have been listed with respect to the overall CPSoSaware architecture and the technical components collection as listed in D1.4 [2]. Moreover, an approach for automating and orchestrating the integration and deployment process of the components, has been presented. During the next months, the integration activities will progress further aiming to be integrated and demonstrated through the project's pilots. The final integrations as manifested through the demonstrators of the project, will reported in the next and final version of this deliverable.

10 References

[1] "D1.2 Requirements and the Use Cases".

[2] "D1.4 : Second Version of CPSoSaware System Architecture".

[3] "D6.4 Preliminary Evaluation and Assessment of CPSoSaware Platform".

[4] [Online]. Available: https://www.jenkins.io/.

[5] [Online]. Available: https://www.nongnu.org/cvs/.

[6] [Online]. Available: https://subversion.apache.org/.

[7] [Online]. Available: git.

[8] [Online]. Available: https://www.mercurial-scm.org/.

[9] [Online]. Available: https://www.jenkins.io/doc/book/pipeline/.

[10] "Preliminary Version of CPSoS Simulation Tools and Training Data Generation".

[11] "D3.1 Algorithms for monitoring the user and analyzing the scene by fusioning multimodal data".

[12] "Wu, Bichen, et al. "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving." Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 2017".

[13] "Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." arXiv preprint arXiv:1602.07360 (2016)".

[14] “ He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.”.

[15] “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”.

[16] “Zhou, Yin, and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018”.

[17] “Yan, Yan, Yuxing Mao, and Bo Li. "Second: Sparsely embedded convolutional detection." Sensors 18.10 (2018): 3337”.

[18] “Graham, Ben. "Sparse 3D convolutional neural networks." arXiv preprint arXiv:1505.02890 (2015)”.

[19] “Rothe, Rasmus, Matthieu Guillaumin, and Luc Van Gool. "Non-maximum suppression for object detection by passing messages between windows." Asian conference on computer vision. Springer, Cham, 2014”.

[20] [Online]. Available: https://github.com/riebl/ros.

[21] [Online]. Available: https://www.asam.net/standards/detail/openscenario/.

[22] [Online]. Available: https://github.com/carla-simulator/traffic-generation-editor.

[23] “D4.8 Final Version of CPSoS Runtime Secutiry Monitoring Approaches”.

[24] “D4.7 Final Version of Design and Implementation of Smart Dynamic Network Structures for Dependable CPSs”.

[25] “D3.5 Modules for enabling Security and Trust”.

[26] “Preliminary Version of Design and Implementation of Smart Dynamic Network Structures for Dependable CPSs”.

[27] “D3.2 OPENCL PROTOTYPE TO SUPPORT DISTRIBUTED EXECUTION OF KERNELS AND DATA TRANSFERS IN CPSS”.

[28] “https://wiki.debian.org/iwconfig#:~:text=iwconfig%20is%20similar%20to%20ifconfig,frequency%2C%20SSID).”.

[29] “https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-networkmanager_tools”.

[30] [Online]. Available: https://man7.org/linux/man-pages/man8/sysctl.8.html.

[31] [Online]. Available: https://www.jenkins.io/.

[32] “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”.

[33] “Lang, Alex H., et al. "Pointpillars: Fast encoders for object detection from point clouds." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019”.