

D1.4 - Final release of the smart city use case

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M29, May 2020 (Due to COVID situation this deliverable has been submitted on M31, July 2020)
Dissemination Level	PU
Nature	R
Author(s)	Danilo Amendola (CRF-MAS) danilo.amendola@crf.it
Contributor(s)	Roberto Cavicchioli (UNIMORE), Luca Chiantone (MODENA), Jorge Montero Gomez (ATOS), Erez Hadad (IBM), Eduardo Quiñones (BSC), Maria A. Serrano (BSC), Mara Tonietti (MAS).
Reviewer(s)	Jorge Montero Gómez(ATOS), Rut Palmero (ATOS)
Keywords	Use cases, connected vehicles, smart city, edge computing, fog computing

Notices: The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No "780622".



© 2018 CLASS Consortium Partners. All rights reserved.

Change Log

Version	Author	Description of Change
V0.1	D. Amendola	Initial Draft
V0.2	Erez (IBM), Roberto (UNIMORE), Maria (BSC)	Contributions to this deliverable
V0.3	D. Amendola	Merged contribution from partners, final version for internal review
V0.4	ATOS	Internal Review
V0.5	D. Amendola	Revised version
V1.0	Maria A. Serrano	Final Version, ready to EC revision

Table of contents

Executive Summary	5
1 Description of the combined big-data analytics workflow implemented on the use cases 6	
1.1 Big-data analytics methods overview.....	6
1.1.1 The CLASS combined workflow	7
1.2 Implementation details of the big-data analytics methods	8
1.2.1 Sensor fusion (1).....	8
1.2.2 Object detection (2).....	10
1.2.3 Object tracking (3)	11
1.2.4 Data deduplication (4).....	11
1.2.5 Trajectory prediction (5).....	12
1.2.6 Pollution computation (6)	13
1.2.7 Data aggregation (7).....	13
1.2.8 Collision detection (8).....	14
1.2.9 Warning Area (WA) generation (9).....	15
1.2.10 Alert visualization (10).....	16
1.2.11 Parking counters (11)	16
1.2.12 Dashboard visualization (12)	16
1.2.13 Predictive models - MATSim (13)	17
1.3 DKB Generation Sub-Workflow	20
1.4 Obstacle Detection sub-workflow	21
1.4.1 Scenario 1: Via Manfredo Fanti - Str. Attiraglio.....	23
1.4.2 Scenario 2. Str. Attiraglio.....	24
1.4.3 Scenario 3. Via Manfredo Fanti (Parking).....	25
1.4.4 Scenario 4. Via Maria Montessori/Via Pico della Mirandola roundabout.....	26
1.4.5 Scenario 5. Parking exit at the Via Pico della Mirandola roundabout.....	28
1.5 Digital Traffic Sign Sub-Workflow	29
1.6 Smart Parking Sub-Workflow	30
2 Description of the Modena Automotive Smart Area (MASA)	31
2.1 Modena Automotive Smart Area (MASA)	31
2.2 Maserati vehicles prototypes	31
2.2.1 Prototype vehicles	31
2.2.2 Sensing components.....	33

2.2.3	Communication components	35
2.2.4	Computation components.....	35
2.2.5	Position and Field of View of the Sensors	36
2.2.6	Vehicle's Software Architecture	36
3	Acronyms and Abbreviations	38
4	References	39

Executive Summary

This deliverable describes the final release of the smart city use cases (UCs). A first version of the UCs has been provided in deliverable D1.2. In the following pages a revised version of the UCs is provided and the current status of implementation is described.

This deliverable describes the technologies used in the CLASS project in order to meet the requirements of the use case defined in the project's DoA. The deliverable is composed of two parts: i) the description and preliminary analysis of the combined real-time big data analytics implemented on the use-case (Section 1); ii) the description of the Modena Automotive Smart Area (MASA), and vehicles populated with IoT sensors (Section 2).

The deliverable contributes to MS3, the target at MS3 is the smart city use case developed with the CLASS architecture, and ready to be evaluated in the MASA.

1 Description of the combined big-data analytics workflow implemented on the use cases

This section describes the *combined big-data analytics workflow* implementing the application responsible of generating of the *Data Knowledge Base (DKB)* from which valuable knowledge is extracted from the city and the connected cars, and the three applications of the CLASS use-cases (obstacle detection, digital traffic sign, and smart parking), built upon the information included in the DKB.

A preliminary evaluation of the end-to-end big-data analytics integration executed on the CLASS software architecture is provided in Deliverable D2.6 [1].

1.1 Big-data analytics methods overview

The combined big-data analytics workflow is composed of the following analytics methods (the numbers of the list are used to identify the methods along the document, i.e. method 1 refers to sensor fusion and so on):

1. **Sensor fusion.** This method is responsible of object identification and computation of its position based on the raw data collected from sensors included in the vehicles.
2. **Object detection.** This method is responsible of object identification and computation of its position based on raw data from the cameras of the city.
3. **Object tracking.** This method tracks the movement of objects identified with the *method 2*, across multiple video frames.
4. **Data deduplication.** This method identifies and removes those objects identified with the *method 3* by multiple cameras of the city.
5. **Trajectory prediction.** This method predicts the trajectory of vehicles and cyclist identified with the *method 3* based on previous tracked positions.
6. **Pollution computation.** This method estimates emissions from the drive cycles of vehicles identified in *method 3* based on the emission class (e.g., Euro 1, Euro 2, Euro 3, etc) and vehicle speed over time.
7. **Data aggregation.** This method is responsible of aggregating all the information generated in the previous analytics methods into the common DKB.
8. **Collision detection.** This method identifies potential collisions, based on the intersection of the trajectory predictions of two objects (computed by *method 5*).
9. **Warning Area (WA) generation.** This method obtains all the information included in the DKB surrounding a connected vehicle.
10. **Alert visualization.** This method visualizes into the connected vehicle the following information: (i) the potential collision of the connected vehicle with other vehicles within the WA, and (ii) the available parking slots in the DKB.
11. **Parking counters.** This method computes the available parking slots based on cameras from the city.
12. **Dashboard visualization.** This method is responsible of visualizing the information included in the DKB.
13. **Predictive models.** This method implements the digital traffic signs application on the MatSim simulator [2].

1.1.1 The CLASS combined workflow

Figure 1 shows the combined big-data analytics workflow as a Direct Acyclic Graph (DAG) in which nodes represent the analytics methods described above (the number in the node identifies the method) and edges represent the data exchange (and so dependencies) among the different analytics. Notice that the order of execution of methods 8 and 9 can be swapped.

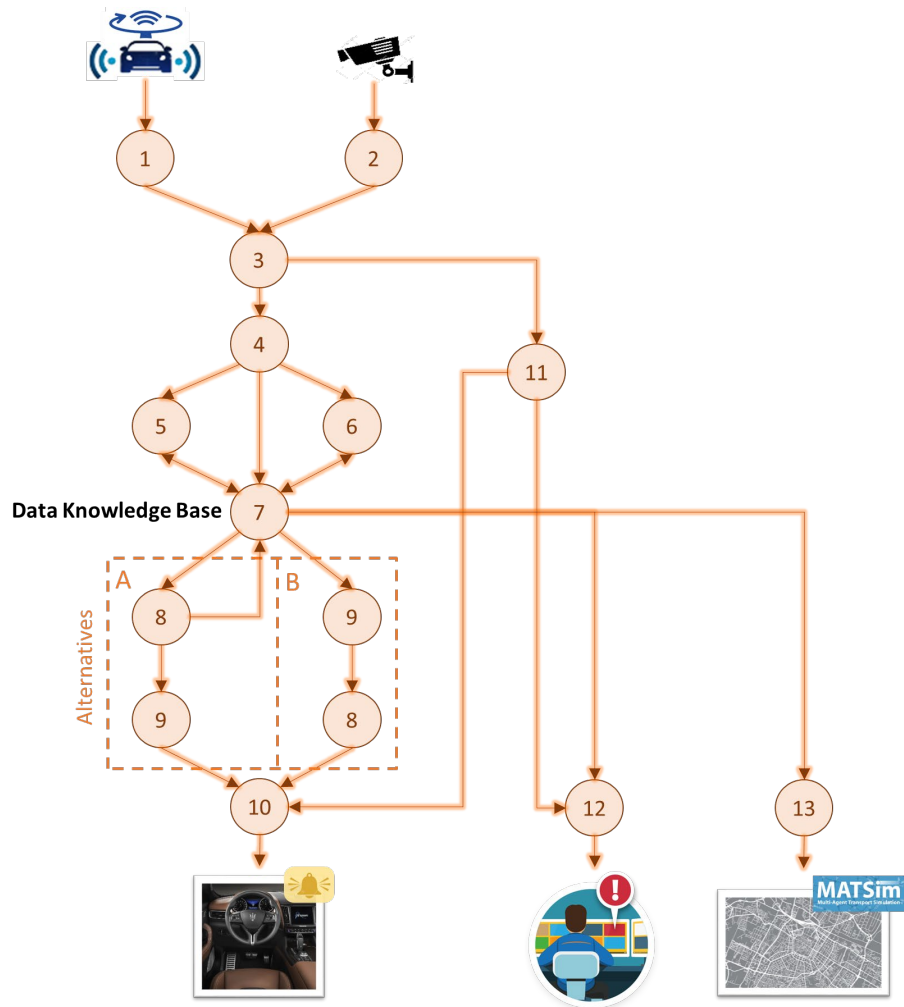


Figure 1: The combined big-data analytics workflow.

Figure 2 shows the same workflow as the one presented in Figure 1, but considering several data-sources; concretely, three city cameras and one connected vehicle are considered as input sources, and two connected vehicles and the city control room are receiving the information. The figure also identifies where data analytics methods can be executed, i.e., in the edge/fog, in the cloud or in both locations, aiming to efficiently distribute the workload across the compute continuum.

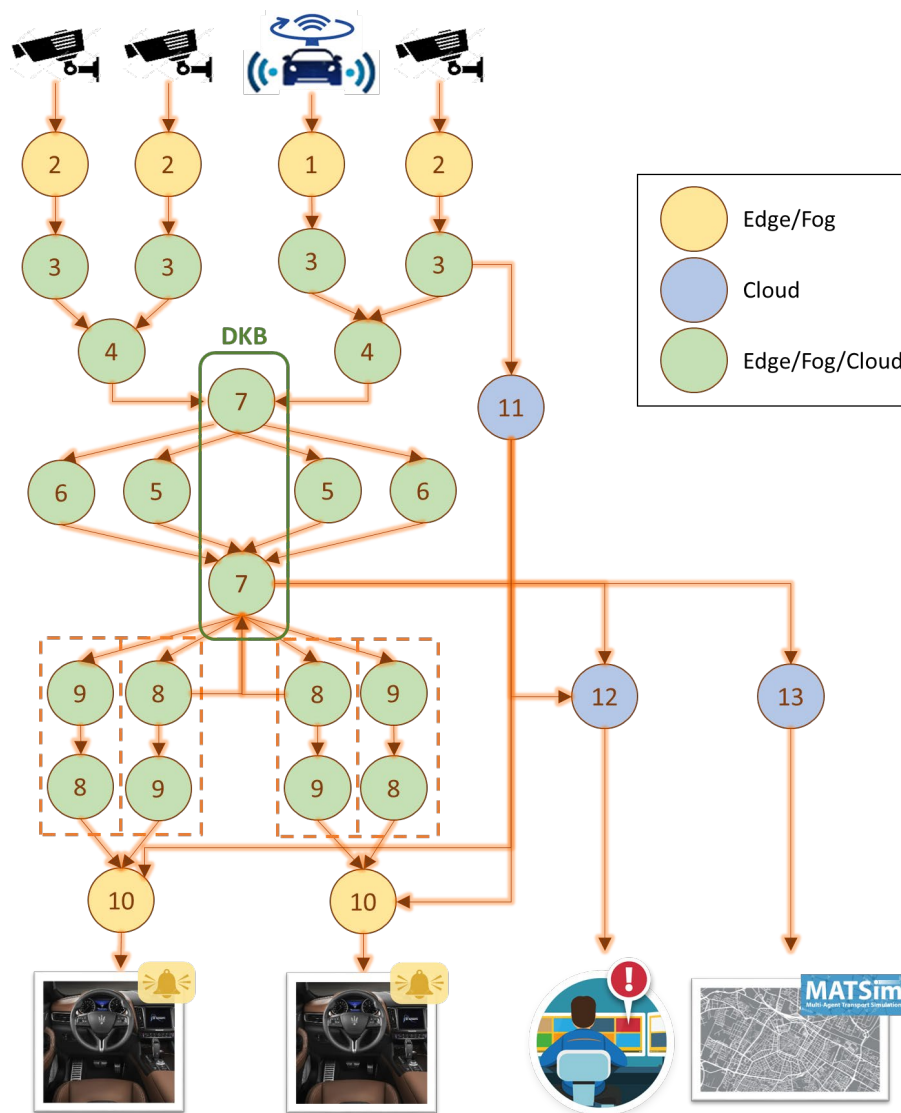


Figure 2: CLASS combined big-data analytics workflow when considering multiple data-sources.

Next Section 1.2 provides a detailed description of the big-data analytics methods. Then, Sections 1.3, 1.4, 0 and 0, describe in detail the sub-workflows that implement the different CLASS applications, i.e., the generation of the DKB, the obstacle detection, the digital traffic sign, and the smart parking, respectively.

1.2 Implementation details of the big-data analytics methods

1.2.1 Sensor fusion (1)

The core tasks for self-driving cars include environmental perception, precise localization, and path planning. To do so, a precise categorization and localization of road-objects, such as cars, pedestrians, cyclists, and other obstacles, is needed. Fusing data from the cameras and LiDARs (Light Detection and Ranging) sensors has become a trend in the AD (Autonomous Driving) field, being a good compromise to obtain good classification and 3D detection. The main problem of this approach is its high computational cost, which prevents its adoption on real-time sensitive scenarios. In the context of 2D camera object detection, Convolutional Neural

Networks (CNN) are often adopted (see data analytics [3] for more information). LiDARs instead produce a 3D point cloud, which is then processed to place 3D bounding boxes (BBs) around the objects via clustering methods. Density-based clustering is the most suitable for processing LiDAR point clouds. The most widely adopted algorithm in this class is DBSCAN (Density-Based Spatial Clustering of Applications with Noise).

We merged camera detection and LiDAR clustering by developing a modified parallel DBSCAN algorithm that exploits the features of LiDAR point cloud, optimizing a YOLO-v3 CNN and deploying an open-source real-time framework that combines camera 2D BBs with LiDAR clustering, as depicted in Figure 3.

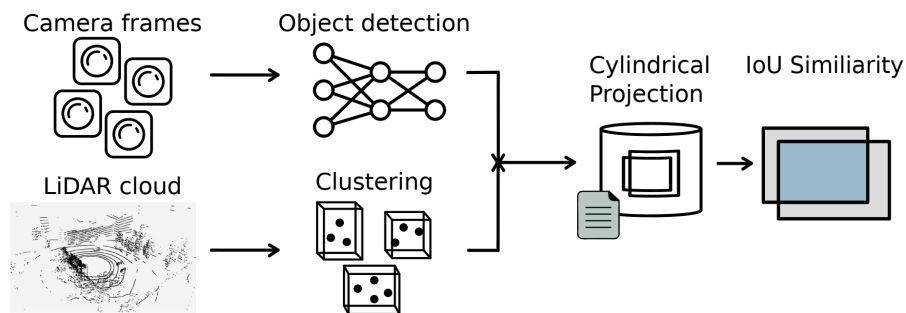


Figure 3: Sensor fusion scheme

For the considered setting, the proposed calibration scheme is depicted in Figure 4. It consists in five phases following the initial intrinsic calibration of the camera.

- 1) *Feature extraction.* The calibration pattern is registered both from camera and LiDAR. From the camera frame, the pattern is extracted via a green color filter and, thanks to Hough transform for circles, the center of the holes is easily computed. From the LiDAR, a range filter is used to find the panel, a Sobel filter to extract the edges, and a conic fitting to select the centers.
- 2) *Cylindrical projection.* The camera calibration matrix is cylindrically projected to obtain a coherent representation with the LiDAR.
- 3) *Cylinders alignment.* The LiDAR cylinder is shifted of a fixed offset (in cm) given by the positioning of the sensors. The camera cylinder is aligned with the LiDAR one by homography, exploiting the centers of the holes found at step 1.
- 4) *Evaluation.* A good calibration is achieved when the vertical edges of the distorted camera frame, projected on the LiDAR depth map using the calculated homography, are parallel and have the same length. If this is not the case, a further correction is needed (step 5). Otherwise, calibration is complete.
- 5) *Correction.* Translation and rotation can be corrected as follows:
 - Translation: define a neighborhood of the sensors positioning displacement to adjust the overlapping, moving the LiDAR cylinder. Go back to step 4.
 - Rotation (roll): from the camera frame pick two points that, projected on the LiDAR cylinder, have the same y value. Check their angle on the camera image with regard to the x-axis: if the angle is not zero, the camera image should be rotated by that angle. Go back to step 2.
 - Rotation (pitch): the perpendicular point of the camera cylinder needs to match the LiDAR one. If these two points have a different height when projected to the horizon, the camera distortion must be fixed. To do so, replace the focal center of

the camera with the point that is aligned with the LiDAR one when projected to the cylinder. Go back to step 2.

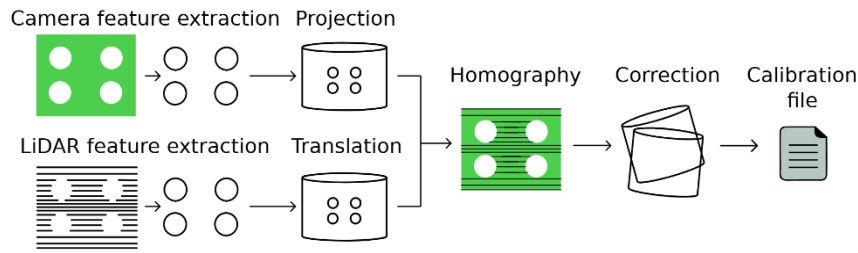


Figure 4: Calibration scheme

For a complete description of the algorithms, please refer to [4]

1.2.2 Object detection (2)

From traditional and smart camera frames, objects are detected and classified with an optimized version of Yolo-v3 that runs on the tkDNN framework. Yolo-v3 is a CNN that is well known and the state of the art for object detection. tkDNN is a Deep Neural Network library built with cuDNN [5] and TensorRT [6] primitives, specifically thought to work on NVIDIA Jetson Boards. Its main goal is to exploit NVIDIA boards as much as possible to obtain the best inference performance. Moreover, a training from scratch has been performed on the BDD100K Berkeley Dataset to better recognize road objects.

After detection and classification, we need to compute the global position of the road user. To do so, each camera is manually calibrated to match known points in the image with their GPS position on a georeferenced map. After initial calibration, the homography matrix obtained is used to project the lower center of the bounding-box (in pixels) onto the GPS plane.

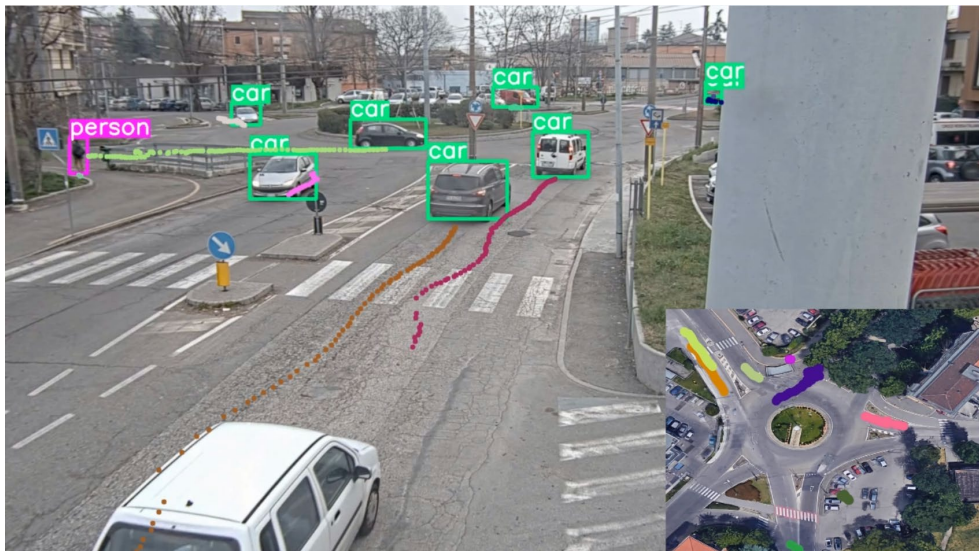


Figure 5: Sample of camera detection coupled with tracking and visualization on eagle-eye map (colors are different between images and map).

The code for the object detection can be found in the CLASS GitHub at <https://github.com/class-euproject/tkDNN>.

1.2.3 Object tracking (3)

Visual tracking algorithms are computationally intensive and not suitable for our real-time scenario. Our idea is to develop a fast method to track objects from pole-mounted cameras, therefore an Extended Kalman Filter (EKF) has been adopted. After our Object detection method (2) detects the bounding-boxes of the objects, the central bottom point of the bounding box is taken as a reference of that object and the tracker is instantiated.

However, the tracker implementation is general, but takes as input a point for an object for each frame and track those objects with an aging mechanism.

We are able, with just the position of the object (x,y) to predict not only the new position (x',y'), but also the velocity v, the yaw ψ , and the yaw-rate $\dot{\psi}$. Hence, the state of EKF is:

$$\begin{bmatrix} x \\ y \\ \psi \\ v \\ \dot{\psi} \end{bmatrix}$$

While the state transition adopted to update the EKF and compute the predicted point is:

$$\begin{bmatrix} x + \frac{v \cdot (-\sin(\psi) + \sin(T \cdot \dot{\psi} + \psi))}{\dot{\psi}} \\ y + \frac{v \cdot (\cos(\psi) - \cos(T \cdot \dot{\psi} + \psi))}{\dot{\psi}} \\ T \cdot \dot{\psi} + \psi \\ v \\ \dot{\psi} \end{bmatrix}$$

This tracking method is fast and embarrassingly parallel, since we can create each tracker as a separate entity and update its state based on the nearest corresponding object detected in a subsequent frame. This allows to compute the tracking on the edge devices in few milliseconds for more than 30 road users detected each frame.

The code for the tracker can be found in the CLASS GitHub at https://github.com/class-euproject/tracker_CLASS.

1.2.4 Data deduplication (4)

When multiple objects are detected in the same area by cameras that share part of their field of view or by a smart connected car moving in the same area, we have to manage the duplicated road users that are detected by the different actors.

Our simple method in this case is to search for all nearest object with the same category of the considered one, compare their position and remove the duplicate ones that are within a threshold value. The threshold is chosen depending on the category of the road user.

The code for the aggregator and de-duplicator can be found in the CLASS GitHub at <https://github.com/class-euproject/deduplicator>.

1.2.5 Trajectory prediction (5)

As presented in D1.2 [3], the trajectory prediction was initially described to be executed in the pCEP (parallel Complex Event Processor) with the collision detection. After discussions to have the best scenario, the computation of the trajectory prediction has been extracted and is executed in collaboration with PyWren [7] (more information about this integration can be found in D5.4 [8]).

The first version described used quadratic regressions to define the parabola and get the predicted X_n and Y_n based on t_n . In this version only one point in the future is predicted, for one object. This had a long way to improve until converge in the actual version.

The actual version has the next list of updates to fit with the requirements of CLASS:

- Compute multiple predictions per object, in different future time points.
- Input samples are not equally apart in time.
- Predict trajectories for multiple objects at the same time.
- A set of parameters to define global properties of the trajectory prediction computation:
 - o $QUAD_REG_LEN = 20$ → maximum number of past trajectory points to manage.
 - o $QUAD_REG_OFFSET = 5$ → number of points to predict.
 - o $QUAD_REG_MIN = 5$ → minimal number of points to calculate the trajectory.
 - o $PRED_RANGE_MIL = 200$ → range for predicted points in milliseconds.

As an example, in Figure 6 is shown the output of the trajectory prediction taken into consideration the first point of the total of $QUAD_REG_OFFSET$ defined with a $PRED_RANGE_MIL$ of 200 milliseconds. They represent the X vs T and Y vs T respectively.

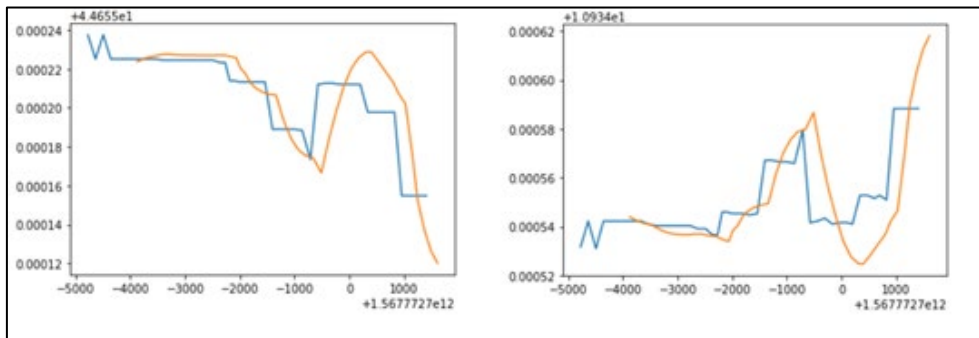


Figure 6: Example of X and Y outputs for the trajectory prediction using a range of 200 milliseconds. In blue the original path; in orange the predicted path.

The core trajectory prediction needs to be applied to each object in the MASA, and only when new location data is available for the object. In the overall CLASS solution, new object data is made available in micro-batches called *snapshots*, with each snapshot pointing to a group of objects whose data is updated. The most efficient response to such a snapshot to compute the trajectory for the included objects concurrently. This is the value of the PyWren integration, which loads the snapshot and then applies a *map()* operation to compute trajectory concurrently on all included objects using OpenWhisk [9] actions. See details in the analytics layer final release report D5.4 [8].

The code for the trajectory prediction can be found in the CLASS GitHub at <https://github.com/class-euproject/trajectory-prediction>.

1.2.6 Pollution computation (6)

The pollution computation method consists of a data emissions model to compute the levels of contamination of the MASA. See description in D1.2 [3].

1.2.7 Data aggregation (7)

The information included in the DKB is structured in the data model presented in Figure 7 and implemented with dataClay [10], including the following (Python) classes:

- *DKB*: It corresponds to the main Python class; it contains a list of *EventsSnapshot*.
- *EventsSnapshot*: It contains the list of objects included in the DKB within a given period of time, e.g., in the last 30 s.
- *Object*: It represents each of the objects included in the DKB, each characterized by:
 - *id_object*: Unique identifier for the object.
 - *type*: Type of object detected, i.e., *person*, *car*, *truck*, *bus*, *motorcycle*, *bike*, *train*.
 - *speed*: Last computed speed of the object.
 - *yaw*: Last object orientation.
 - *trajectory_p[x,y,t]*: Prediction of the trajectory of the object, including the future positions (in GPS coordinates) for a given time, computed with the *method 5*.
 - A list of *Event*.
- *Event*: It represents the different positions of an object for a given period of time, including the following information:
 - *id_event*: Unique identifier for the event.
 - *timestamp*: The time at which the position of the object has been recorded
 - *latitude_pos*, *longitude_pos*: GPS coordinates of the object at the *timestamp*.

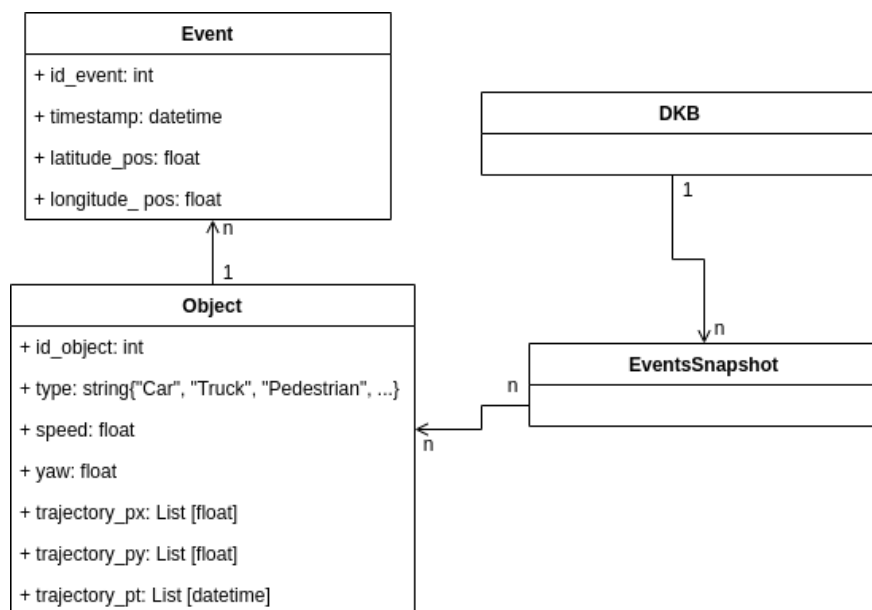


Figure 7. Diagram of the data model of the DKB.

Figure 8 shows the evolution of the DKB when two cars (blue and green) are identified for a given time slot. At *EventsSnapshot 1*, the blue car Car_b has been detected and only one event

with its GPS position pos_{b1} and time stamp $timeS_{b1}$ is stored. At *EventsSnapshot 2*, the same car is detected and then, tracked, so speed $Speed_b$ and yaw Yaw_b can now be computed. Similarly, the green car is also detected and tracked starting at *EventsSnapshot 3*. The trajectory prediction of both cars, $Trajec_b$ and $Trajec_g$, are computed at *EventsSnapshot 4* and *EventsSnapshot 5*, respectively.

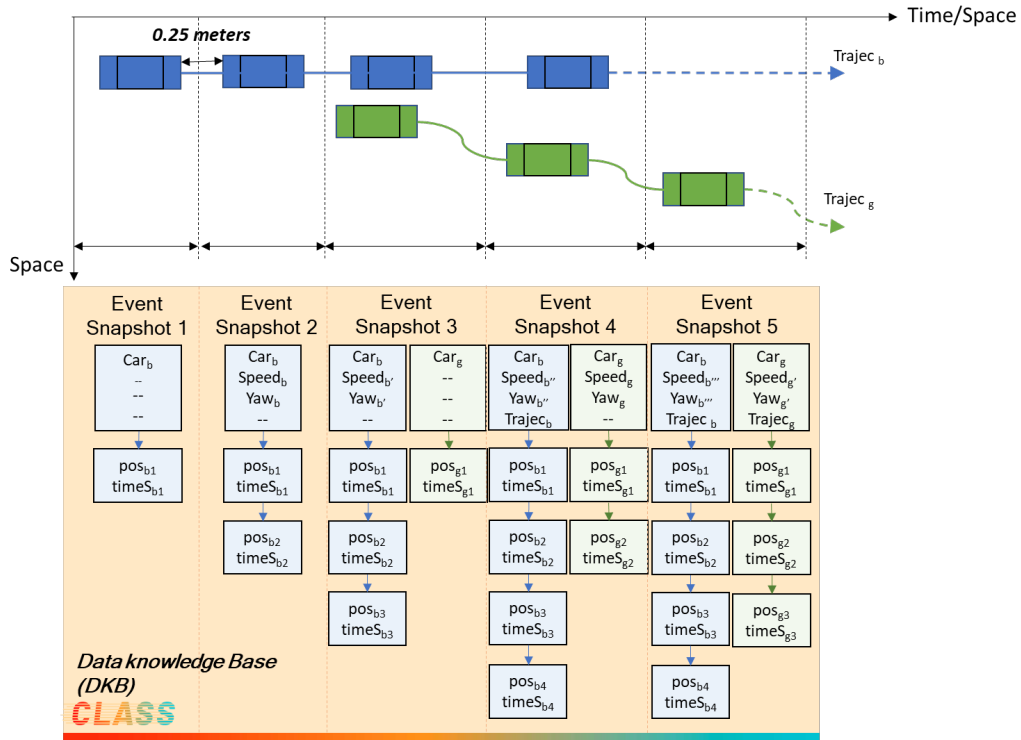


Figure 8. Example of CLASS Data knowledge base (DKB).

The code for the data aggregation can be found in the CLASS GitHub at <https://github.com/class-europroject/dataclay-class>.

1.2.8 Collision detection (8)

The core collision detection algorithm is applied to a pair of street objects (car, pedestrian, etc) from the DKB. These objects need to have their trajectories predicted in a previous workflow step as described above. The collision detection then computes predicted path intersections as following:

- A quadratic regression is computed to get the functions for the two objects that are in the scope, to later get possible intersections. This is done using the same approach presented in the trajectory computation component.
- When the two trajectory functions are defined, both are crossed to get the intersection points, if exists. The intersections could be get using linear or non-linear interpolation. Figure 9 shows an example of the intersection of two objects, and another example without intersections, respectively.

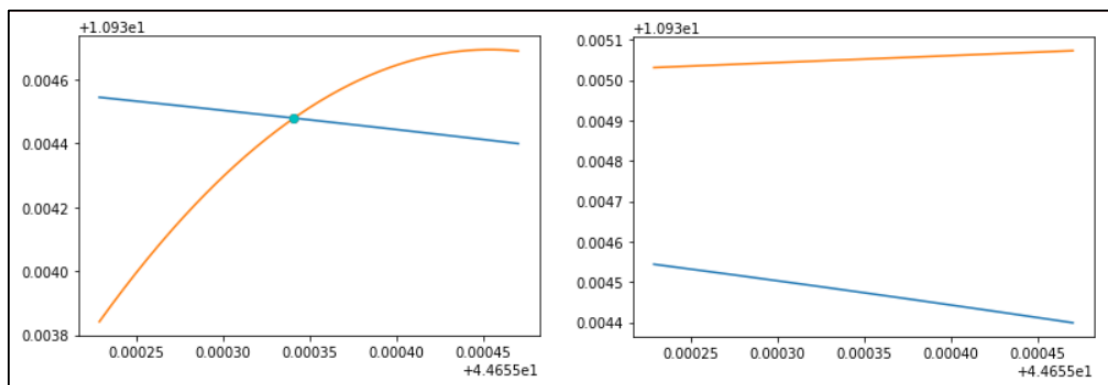


Figure 9. Trajectories for different objects with two possible options: (left) with intersection; (right) without intersection

- The next step is to get the timestamps for both objects when they get the intersection point. At this phase, a variable of 2 seconds has been defined to arise an alert if they converged at the intersection point with a difference of less than 2 seconds.
- At the end, if potential collisions are detected, they are sent with the information of the two objects that could crash, the X and Y coordinates and the timestamp associated. Figure 10 shows an example of the data of a potential collision alert.

```

WARNING!!! Possible collision detected
v_main: 2128793555 v_other: 1348026716
x: 44.65553424126951 y: 10.934492820873857 t: 1567772745174.0

```

Figure 10. Example of an alert when a potential collision is detected

- Those collisions are finally sent to the next component of the workflow, in our case to the WA alert visualization.

Taken into consideration the solution described, there is room to improve by applying different analytics to obtain the intersections, and by adding more parameters to the computation part, as the size of the objects, to perform better potential collisions (analyzing the Risk Assessment presented in [11]).

Additional detailed are described in D5.4 [8].

The code for the collision detection can be found in the CLASS GitHub at <https://github.com/class-euproject/collision-detection>.

1.2.9 Warning Area (WA) generation (9)

Within the overall workflow, collision detection is intended to be applied between a car and objects within its *warning area*. This is where the PyWren map/reduce model is integrated with both warning area computation and collision detection to provide an efficient concurrent solution. Upon request / event identifying a specific car object, the PyWren application filters DKB objects that belong in the car's warning area based on a geometric filter. For each object that passes the filter (i.e., belongs in the warning area of the given car), collision detection is applied between that object and the car. The filter and collision detection are applied concurrently on all objects using a map() operation.

1.2.10 Alert visualization (10)

We developed a lightweight 3D user interface that will show to the driver its position in the MASA, the position of the road users detected by the car and by the infrastructure and, finally, the warning about possible collisions.

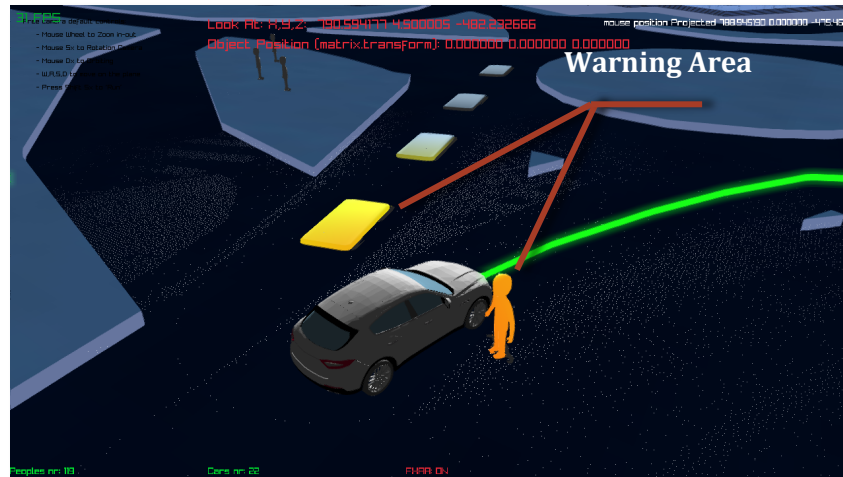


Figure 11 The 3D viewer with colored road users for warning.

The color of the different objects changes from bright orange when close to the car to white when far enough. Warning signals for possible collisions are shown as icons on top of the interface.

Other characteristics of this user interface can be found in [12]

1.2.11 Parking counters (11)

The parking area behind the Modena central railroad station has only one entrance and one exit. Both paths are covered by our traditional cameras. With our detection and tracking algorithm, we are able to count ingoing and outgoing vehicles. Given the maximum size of the parking lot, we maintain a global counter at fog/cloud level so that, when a vehicle requests a parking slot, the CLASS software architecture can answer the request.

This implementation requires only two traditional cameras instead of a large number of dedicated sensors for each car slot in the parking area.

1.2.12 Dashboard visualization (12)

The visualization of the aggregated data of the DKB is possible through an application developed in Python with the PySpark [13] and Folium [14] packages. The PySpark allows connection to an Apache Spark distributed DBMS to recover the DKB aggregated data, while Folium is used to plot the data on a leaflet map, which is able to import tilesets from OpenStreetMap.

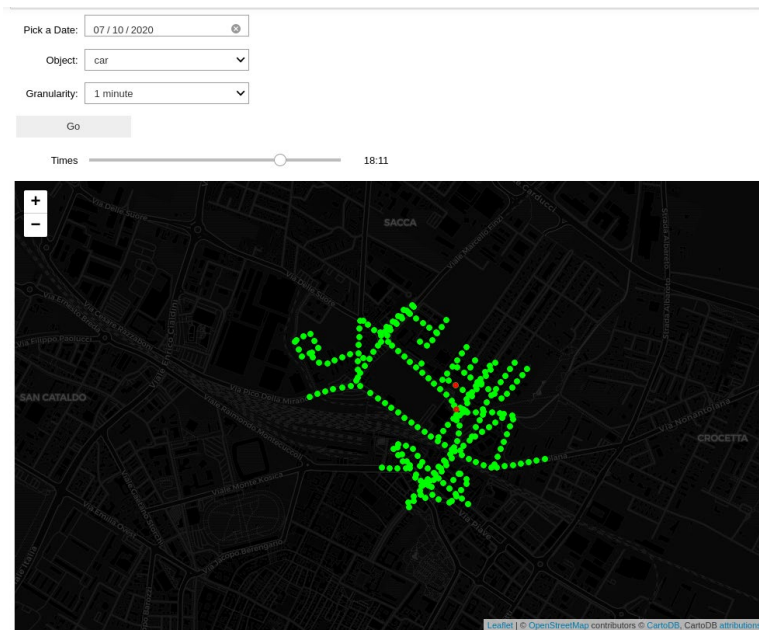


Figure 12: Dashboard visualization.

The code for the dashboard visualization can be found in the CLASS GitHub at <https://github.com/class-euproject/Dashboard-visualization>.

1.2.13 Predictive models - MATSim (13)

We are interested in understanding the potential to improve routing and travel times for emergency services while responding to traffic accidents. Trivially, modifications to real world urban viability must be carefully planned and tested before their actual deployment. We therefore present an extension based on the well-known MATSim simulator[2] for urban transportation that takes inspiration from the Multi-Agent System (MAS) literature. The baseline version of MATSim is only capable of simulating road users that cannot communicate to nearby cars or to the surrounding infrastructure. In contrast, the MATSim extension we present is able to simulate the interactions among sets of connected vehicles within a Smart City.

Our Smart City MATSim extension is composed of multiple modules to simulate cameras, sensors, servers and communication systems so that IoT-equipped vehicles are able to react to unexpected events such as traffic accidents. More specifically, emergency vehicles can communicate with the city infrastructure so to suggest rerouting to the connected vehicles, hence, drastically reducing traffic congestion that occurs within the ambulances' paths. To prove the validity of our extension and related traffic models, we present an extensive set of experiments aimed at assessing the emergency vehicles' response time when moving from/to the accidents' locations.

MATSim [3] (Multi Agent Transport Simulation) is a multi-agent simulation software for traffic management in urban cities. Each agent is a software autonomous entity that represents a pedestrian, a passenger vehicle, an emergency vehicle or a public transportation vehicle.

The interested Smart City area is represented as a network Γ of connected nodes: formally, this is represented as graph $G = (V; E)$, in which vertices represent traffic intersections, and

edges are the links representing the connecting roads. The graph is directed so that we are able to simulate the permitted flow directions. In our MATSim extension, we were able to enhance the baseline urban area representation by adding turning bans and one-way streets in accordance to the really existing area we elect for our simulation. For each link in the road network we associate a FIFO queue, so that whenever an agent enters that link, it is automatically inserted in the corresponding queue. It is trivial to understand that a link characterized by an accident will soon fill its queue, causing a congestion that will reduce or even set to zero the speed of the involved vehicles.

Several kinds of agents are present in our model and each agent has a starting location, one or more activities (in terms of places to be in specific time intervals), and an ending location. More formally, to each agent A we associate a path $P \subset \Gamma$.

The classic agent is the MATSim standard road users that is unable to communicate with the other vehicles or with the surrounding infrastructure. Such an agent can be aware of a traffic accident only when it is located in close proximity to the accident's location. In our model, the classic agent further specializes to be either an autochthonous or an allochthonous road user. The first sub-type describes the behaviour of a road user that knows the city well: hence, in case of a traffic accident it is able to find an optimal rerouting strategy without being helped by the Smart City infrastructure. The optimal route is computed by exploiting the Dijkstra algorithm on the road network, provided that the accident link has been removed from the road network. The other sub-type will attempt to find alternative routes by randomly try nearby roads. More specifically, if an allochthonous agent with a specified ending destination sees an accident at location $siteOfAccident$, a list of connected links from that site is composed as $listOfConnectedLinks$.

A random outgoing link is therefore extracted from that list, unless the only outgoing link is the accident's site. If that is not the case, the agent will move towards the extracted link and checks if its original route ($originalPath$) belongs to its original route to destination. In this case, the rerouting plan is complete. Otherwise this process is iterated.

```

if  $destination == siteOfAccident$  then
  | enqueue( $siteOfAccident$ )
  | return
end
while  $true$  do
  | if  $siteOfAccident$  is the only member of  $listOfConnectedLinks$ 
  | | then
  | | | enqueue( $siteOfAccident$ )
  | | | return
  | | end
  | | else
  | | |  $agentPosition \leftarrow link \leftarrow pickRnd(listOfConnectedLinks)$ 
  | | | update( $listOfConnectedLinks$ )
  | | | if  $link \in originalPath$  then
  | | | | return reroutingPlan
  | | | end
  | | end
  | end
end

```

Figure 13: Allochthonous agent rerouting plan

The smart agent represents a road user driving an ADAS-capable vehicle, hence able to communicate with other ADAS vehicles and the city smart infrastructure. More specifically, this kind of vehicle informs a central authority about its route and this central authority is able to immediately inform the agent that an accident occurred in a link within the route previously communicated by the agent. The agent is therefore able to compute the optimal route even if that accident is far from the agent line of sight.

The emergency vehicle represents an agent acting on behalf of a critical emergency response organization, e.g. police, fire brigade or ambulance. In this work, we aim to prioritize their ability to move within the city during accidents. An emergency vehicle entering a queue of a link is modelled as an occupant of a *Seepage queue*. This allows emergency response vehicles to seep through the queue of both smart and classic vehicles, instead of lining up. Also, we assume that this agent can communicate with other smart agents and with the city infrastructure.

Even if a smart vehicle can immediately reroute as soon as a central authority is aware of an accident, V2V communication can be exploited to avoid smart agent routes to conflict with the emergency vehicle route to/from the accident site. Given that the path of the emergency vehicle (emergencyPath) is composed of N links, we define a Look Ahead Window (LAW) of links represented by the next M links within the emergency agent path starting from its current position. The LAW is communicated to the nearby smart agents, so that they can compare this newly received list of links with a subset (P^*) of their current plan (P); such a subset is composed of the first K subsequent element starting from the current position of the smart agent. If a potential conflict is detected, then the smart agent will reroute (i.e. by removing the overlapping links and recompute Dijkstra).

Experiments have been made over a street network that corresponds to the representation of the MASA. We used several plugins to import to MATSim the original map from the OpenStreetMap database and then we augmented the network with additional information such as capacities for the road links, turning bans and parking spaces.

In this set of experiments, we are interested in observing how fast an ambulance (modelled as an emergency response agent) can travel to/from the site of an accident within the MASA. It is trivial to assume that ambulances travel time will be influenced by the percentage of smart agents over the total number of road users involved in our simulations. Considering this ratio as an experiment parameter is also useful for studying urban viability in the transitional period from exclusively human-driven vehicles to exclusively ADAS vehicles. Since, this transition needs time to complete, it is reasonable to assume that both smart and non-smart vehicle will coexist until next-generation cars will completely overtake the classic transportation means.

	Smart Agent %	Ambulance Travel time	% difference from Scenario 1
Scenario 1	0	109	-
Scenario 2	33	102	6.4
Scenario 3	50	89	18.3
Scenario 4	66	80	26.6
Scenario 5	100	69	36.7

Table 1: Ambulance average travel times with different kind of agents

In Table 1 we see that the average response time of the emergency vehicles dramatically improves when the percentage of smart agents exceeds the 50% over the total number of road users. If all agents were representing ADAS-capable vehicles, decrease in response time would amount to 36.7%.

1.3 DKB Generation Sub-Workflow

The information included in the Data Knowledge Base (DKB) is the basis upon which the three CLASS use-case applications are built. The DKB sub-workflow receives data from the car and city sensors, and it is composed of the big-data analytics methods 1 to 7 and 12, highlighted in Figure 14, and described in previous Section 1.2.

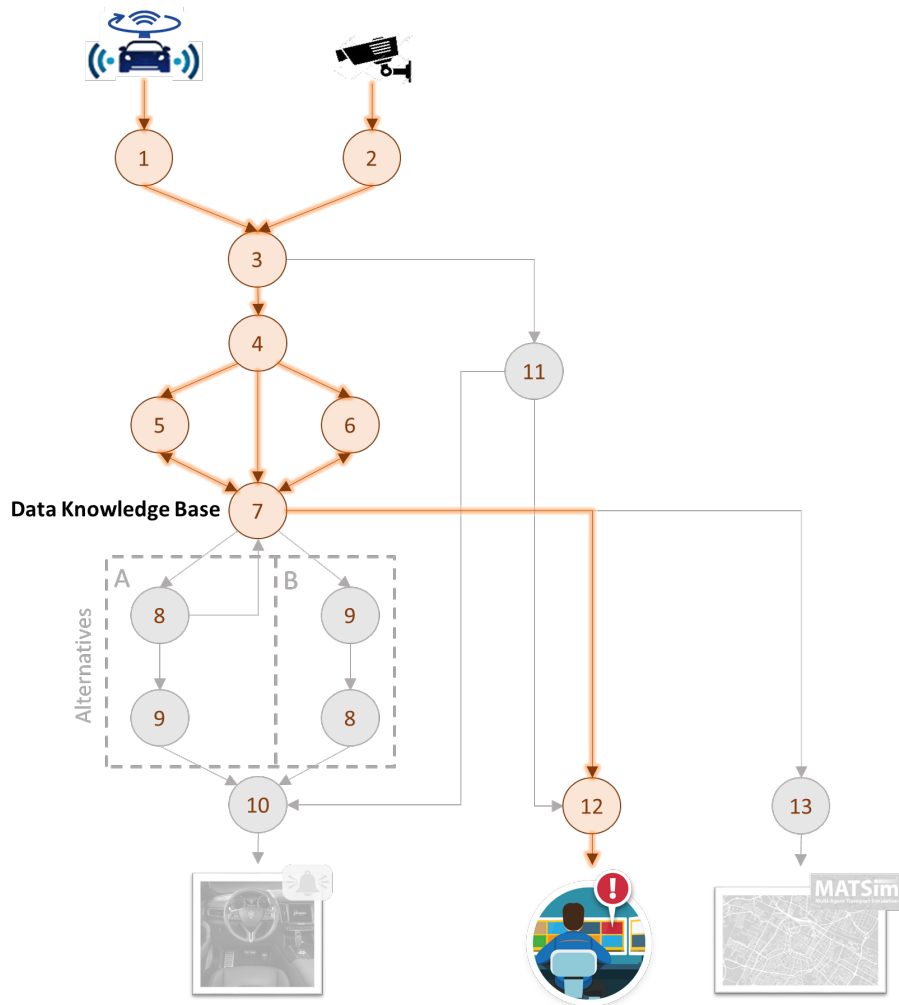


Figure 14: Data knowledge base (DKB).

1.4 Obstacle Detection sub-workflow

The obstacle detection sub-workflow is built upon the information included in the DKB, and it is composed of big-data analytics method 8, 9 and 10, highlighted in Figure 15, and described in Section 1.2.

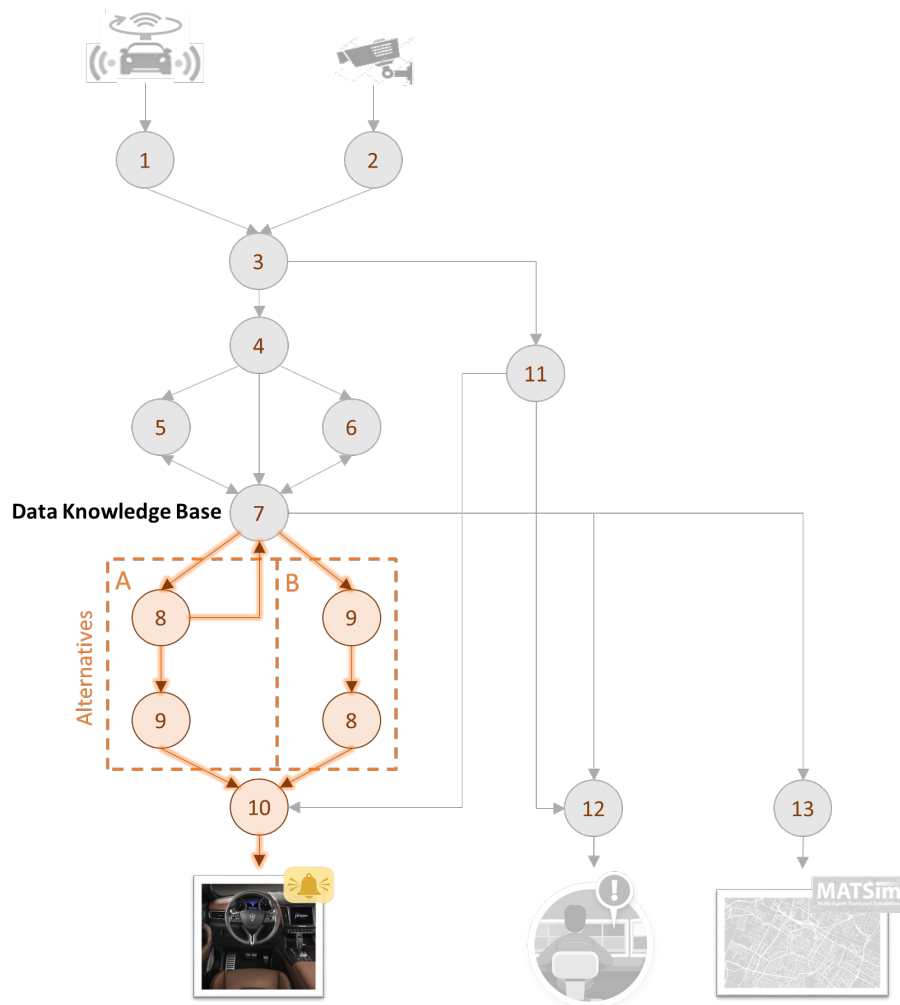


Figure 15: Obstacle detection sub-workflow.

The obstacle detection sub-workflow aims to implement two advanced driving assistant functionalities with the objective of alerting the driver when an object may potentially collide with its vehicle. To do so, the obstacle detection sub-workflow will access to the DKB information surrounding the vehicle, i.e., the Warning Area, containing the information from other vehicles and city sensors. The two functionalities are:

- **Virtual mirror:** This functionality increases the field of view of the vehicle beyond its actual vision, allowing to see “*behind the corner*”.
- **Two sources of attention:** This functionality alerts the driver when he/she has to pay attention to two events in opposite directions.

Table 2 identifies the five demonstration scenarios of the *Obstacle Detection* sub-workflow to be deployed in the MASA, specifying the location (see map in Figure 16), the actors involved in each of the scenarios (see legend in Figure 17) and the expected functionality to be shown.

Table 2. Description of the demonstration scenarios.

Scenario	Location in MASA (see Figure 16)	Actors (see Figure 17)	Functionality
1	Via Manfredo Fanti - Str. Attiraglio	City camera, Smart Car (SC), Connected Car (CC), and Non- Connected Car (!CC)	Virtual mirror
2	Str. Attiraglio	Two City cameras, Smart Car (SC), Truck, and Pedestrian	Virtual mirror
3	Via Manfredo Fanti (at the parking exit)	Two Smart Cars (SC), Connected Car (CC), and Cyclist	Virtual mirror
4	Via Maria Montessori - Via Pico della Mirandola (at the roundabout)	Two City cameras, Smart Car (SC), and Cyclist	Two sources of attention
5	Via Pico della Mirandola (at the parking exit)	City camera, Connected Car (CC), Non-Connected Car (!CC), and Pedestrian	Two sources of attention



Figure 16: Location of the five demonstration scenarios within the MASA.

The number of demonstrators that will be conducted concurrently (and so exploiting the compute continuum) will depend on the number of Maserati vehicles available:

- If two are available, **three** concurrent scenarios will be conducted, i.e., 1 or 2 or 3, 4 and 5.
- If three are available, **four** concurrent scenarios, will be conducted, i.e., 1 or 2, 3, 4 and 5.

Next sections described in detail each of the scenarios.

Figure 17, shows a legend of icons used into the following figures to describe the timeline on each scenarios.

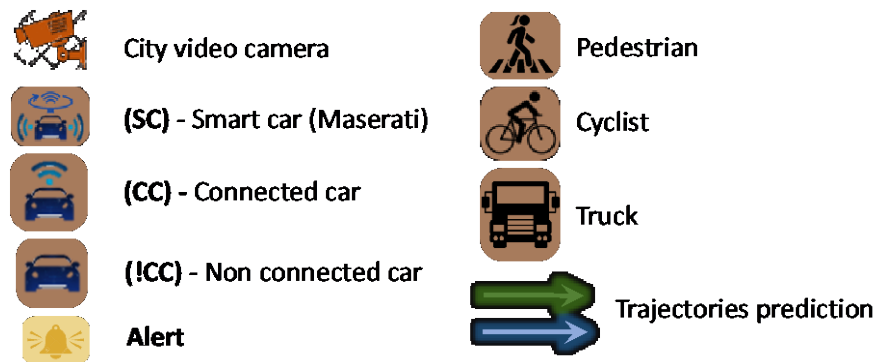


Figure 17: Legend of the icons used to describe the demonstration scenarios.

1.4.1 Scenario 1: Via Manfredo Fanti - Str. Attiraglio

Scenario 1 is described in Figure 18. The actors involved are: a city camera at Attiraglio street that detects a vehicle (!CC), and a Maserati car (SC) that detects a connected car (CC) at Via Manfredo Fanti, both reaching the intersection. The connected car (CC) will receive an alert

due to a hazard detection, since the non-connected car will exit Attiraglio street. Table 3 summarizes the sensors, big-data analytics and services involved in this scenario.



Figure 18: Aerial view of scenario 1 at Via Manfredi Fanti - Str. Attiraglio crossing.

Table 3. Description of Scenario 1

Sensors used for data collection	<ul style="list-style-type: none"> - City camera detecting the (!CC) vehicle - The (SC) for detecting the (CC) vehicle
Data-analytics methods applied	<ul style="list-style-type: none"> - (1) Sensor fusion for detecting (!CC) - (2) Object detection for (CC) - (3) Tracking of (CC) and (!CC) - (5) Trajectory prediction of (CC) and (!CC) - (7) Data aggregation - (8) Collision detection for (CC) - (9) Generation of WA for (CC) - (10) WA alert visualization for (CC)
Services requests to the city (via 4G)	<ul style="list-style-type: none"> - (CC) requests the warning area (WA) generation, based on its own GPS position.

1.4.2 Scenario 2. Str. Attiraglio

Scenario 2 is described in Figure 19. The actors involved are: two city cameras at Attiraglio street that detect a pedestrian, a Maserati car (SC), and a truck that hides the view of the SC driver. The driver will receive an alert due to a hazard detection, since the pedestrian is

crossing the street. Table 4 summarizes the sensors, big-data analytics and services involved in this scenario.



Figure 19: Aerial view of the Scenario 2 at Str. Attiraglio

Table 4: Description of Scenario 2

Sensors used for data collection	<ul style="list-style-type: none"> - City cameras detecting the Pedestrian crossing, the (SC) and the Truck - (SC) providing its own position, direction and speed
Data-analytics methods applied	<ul style="list-style-type: none"> - (1) Sensor fusion for (SC) - (2) Object detection for Pedestrian, Truck and (SC) - (3) Tracking of Pedestrian, Truck and (SC) - (4) Data deduplication from city cameras and (SC) - (5) Trajectory prediction of Pedestrian, Truck and (SC) - (7) Data aggregation - (8) Collision detection for (SC) - (9) Generation of WA for (SC) - (10) WA alert visualization for (SC)
Services requests to the city (via 4G)	<ul style="list-style-type: none"> - (SC) requests the warning area (WA) generation, based on its own GPS position.

Notice that scenarios 1 and 2 cannot occur simultaneously (independently of the number of **(SC)** used, because the truck used in Scenario 2 for obstructing the view of the zebra crossing, may obstruct the detection of the **(ICC)** in the Scenario 1 as well.

1.4.3 Scenario 3. Via Manfredo Fanti (Parking)

Scenario 3 is described in Figure 20. The actors involved are: a Maserati car (SC 1) exiting the parking; and a second Maserati car (SC 2) that detects a connected car (CC) at Via Manfredo Fanti, and a cyclist, the three of them reaching the intersection with the parking exit. Both,

the SC 1 and the CC, will receive an alert due to various hazard detection. Table 5 summarizes the sensors, big-data analytics and services involved in this scenario.

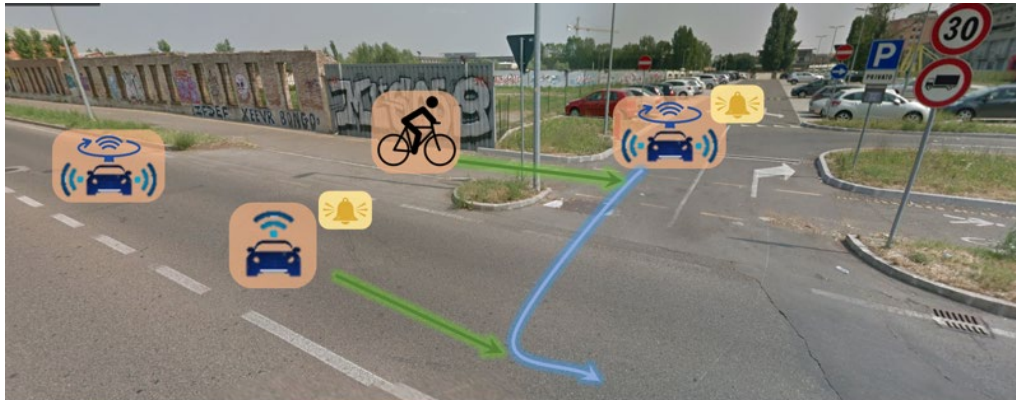


Figure 20: Aerial view of the Scenario 3 at Via Manfredo Fanti

Table 5: Description of Scenario 3

Sensors used for data collection	<ul style="list-style-type: none"> - (SC 1) exiting the parking, providing its own position, direction and speed. - (SC 2) on via Manfredo Fanti, detecting the (CC) and the Cyclist.
Data-analytics methods applied	<ul style="list-style-type: none"> - (1) Sensor fusion for (SC 1) and (SC 2) - (3) Tracking of (SC 1), (CC) and Cyclist - (4) Data deduplication from city cameras and (SC) - (5) Trajectory prediction of (SC 1), (CC) and Cyclist - (7) Data aggregation - (8) Collision detection for (SC 1), and (CC) - (9) Generation of WA for (SC 1), and (CC) - (10) WA alert visualization for (SC 1), and (CC)
Services requests to the city (via 4G)	<ul style="list-style-type: none"> - (SC 1) and (CC) request the warning area (WA) generation, based on its own GPS position.

1.4.4 Scenario 4. Via Maria Montessori/Via Pico della Mirandola roundabout

Scenario 4 is described in Figure 21. The actors involved are: a connected car (CC) reaching the roundabout; and two city cameras that detect the CC and a Cyclist at the roundabout. The CC will receive an alert due to a hazard detection, since the cyclist may be crossing the street. Table 6 summarizes the sensors, big-data analytics and services involved in this scenario.



Figure 21: Aerial view of the Scenario 4 at the Via Maria Montessori/Via Pico della Mirandola roundabout

Table 6: Description of Scenario 4

Sensors used for data collection	- City cameras detecting the (CC) vehicle and the Cyclist ¹
Data-analytics methods applied	<ul style="list-style-type: none"> - (2) Object detection for (CC) and Cyclist - (3) Tracking of (CC) and Cyclist - (4) Data deduplication from city cameras - (5) Trajectory prediction of (CC) and Cyclist - (7) Data aggregation - (8) Collision detection for (CC) - (9) Generation of WA for (CC) - (10) WA alert visualization for (CC)
Services requests to the city (via 4G)	- The (CC) requests the warning area (WA) generation, based on its own GPS position.

¹The smart camera on the left (Figure 21) is not yet installed due to the COVID-19 outbreak, see details in Section 2.

1.4.5 Scenario 5. Parking exit at the Via Pico della Mirandola roundabout

Scenario 5 is described in Figure 22. The actors involved are: a connected car (CC) exiting the parking, and incorporating to the roundabout; a city camera detecting a non-connected car (!CC) and a pedestrian. The CC will receive an alert various hazard detection. Table 7 summarizes the sensors, big-data analytics and services involved in this scenario.



Figure 22: Aerial view of the Scenario 5 at the Parking exit at the Via Pico della Mirandola roundabout.

Table 7: Description of Scenario 5

Sensors used for data collection	- City camera detecting the (CC) , the (!CC) , and the Pedestrian
Data-analytics methods applied	- (2) Object detection for (CC) , (!CC) , and Pedestrian - (3) Tracking of (CC) , (!CC) , and Pedestrian - (5) Trajectory prediction of (CC) , (!CC) , and Pedestrian - (7) Data aggregation - (8) Collision detection for (CC) - (9) Generation of WA for (CC) - (10) WA alert visualization for (CC)
Services requests to the city (via 4G)	- The (CC) requests the warning area (WA) generation, based on its own GPS position.

1.5 Digital Traffic Sign Sub-Workflow

The digital traffic sign sub-workflow is built upon the information included in the DKB, and it is composed of big-data analytics method 13, highlighted in Figure 23, and described in Section 1.2.

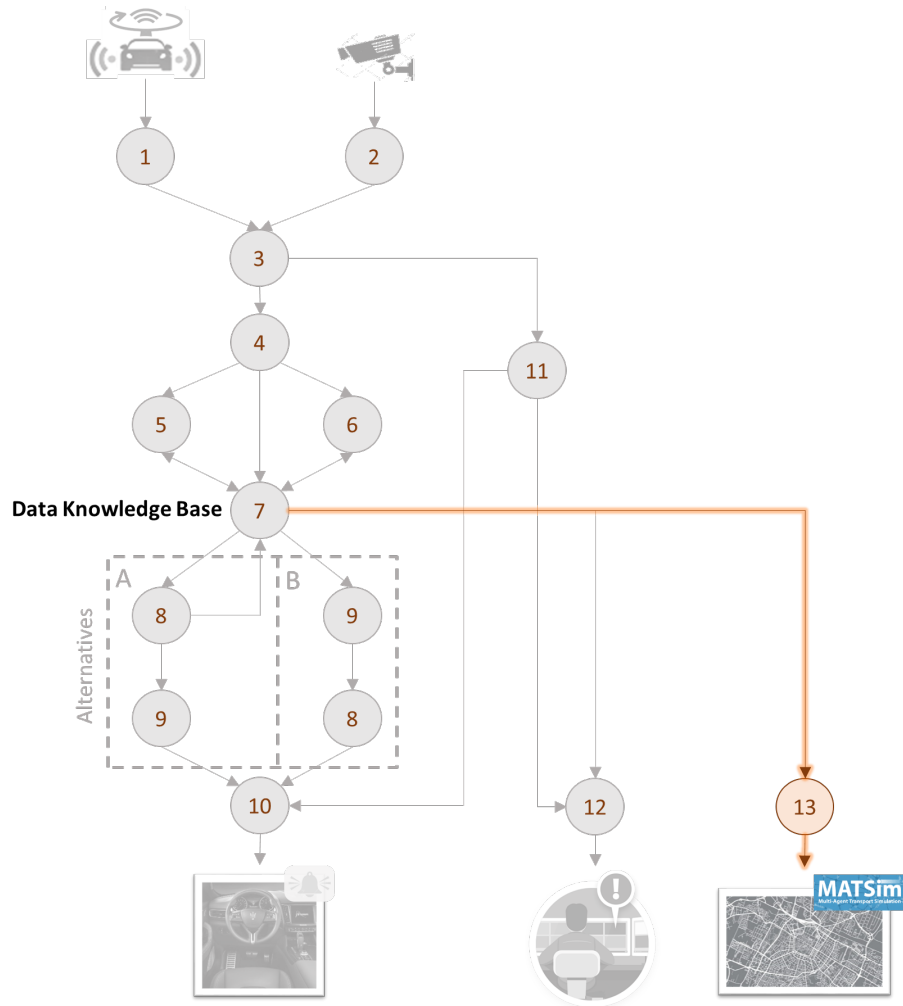


Figure 23: Digital traffic sign sub-workflow.

1.6 Smart Parking Sub-Workflow

The smart parking sub-workflow is composed of big-data analytics method 11, highlighted in Figure 24, and described in Section 1.2.

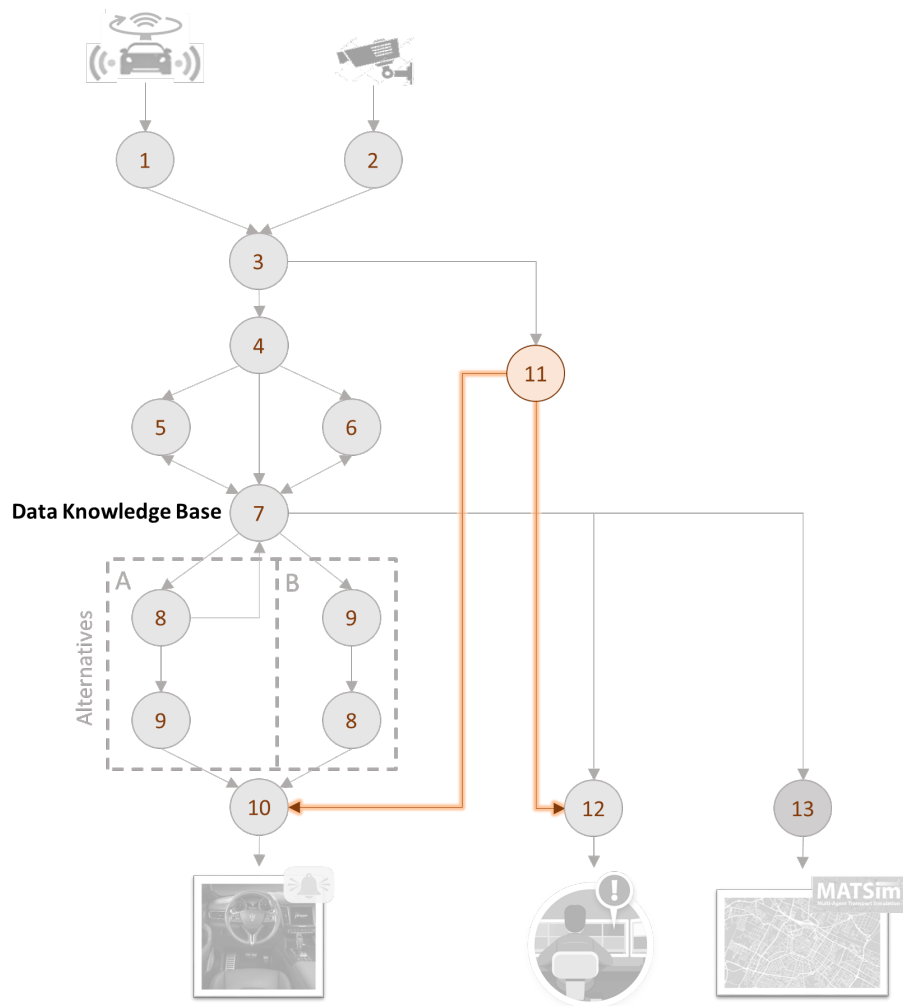


Figure 24: Smart parking sub-workflow.

2 Description of the Modena Automotive Smart Area (MASA)

2.1 Modena Automotive Smart Area (MASA)

The MASA set of sensors has not changed with respect to the initial release of the project described in D1.1 [15]. The smart cameras have not yet been deployed as of M31 of the project due to the COVID-19 outbreak. Nevertheless, we plan to install them by M33. The fog nodes have also been installed inside the old data center of the municipality since the construction work for the new one has been delayed.

2.2 Maserati vehicles prototypes

Within the CLASS project, Maserati has provided three prototype vehicles that incorporate all the sensing infrastructure as will be illustrate in the following, on-board equipment is comprehensive of communication, sensing and computation infrastructure needed to develop the CLASS use cases.

CLASS has three available prototype vehicles: (1) Maserati Quattroporte (MY18), (2) Maserati Levante and (3) Maserati Quattroporte (MY19). In this section we provide a description of each vehicle describing the (i) prototype vehicle, and the on board components: (ii) sensing, (iii) communication and, (iv) computation respectively.

2.2.1 Prototype vehicles

Prototype 1: Maserati Quattroporte (MY18)

The first vehicle is a Maserati Quattroporte, a four-door full-sized luxury sports sedan vehicle (F segment). Specifically the model provided to the project is a Model Year 18, with a gasoline engine V6, 3.0l, 330hp. Figure 25 shows an image of the car with the livery of the Class project and the European Commission.



Figure 25: Prototype 1, Maserati Quattroporte model year 2018

The first car was available since the first day of the project and all the components (view 2.2.2) have already been installed and integrated for the development of the use-cases.

Prototype 2: Maserati Levante (MY19)

The second vehicle is a Maserati Levante, a mid-size luxury crossover SUV (Sport Utility Vehicle). In particular, the Levante acquired for the Class project is a Model Year 19, with a gasoline engine V6, 3.0l, 430hp. Figure 26 shows a picture of the vehicle with the project and EC livery.



Figure 26: Prototype 2, Maserati Levante model year 2019

The second car was provided to the project in September 2018 and all the components (view2.2.2) have already been installed and integrated for the development of the use-cases.

Prototype 3: Maserati Quattroporte (MY19)

The third vehicle is also a Maserati Quattroporte. Specifically the model provided to the project is a Model Year 19, with a gasoline engine V6, 3.0l, 330hp. Figure 27 shows a sample picture of the car.



Figure 27. Prototype 3, Maserati Quattroporte model year 2019 (sample picture)

The third car was provided to the project in June 2020 and the purchase of the sensors is on-going.

For both Maserati Quattroporte, a support for the LiDAR was built and installed on the vehicles roof.

Full ADAS Optional

All the three vehicles incorporate a full Advanced Driving Assistant Systems (ADAS) optional package, which includes the following features:

- Adaptive Cruise Control (ACC) with Stop & Go Function. Automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead, based on a sensor data fusion from RADAR and a front facing camera to detect vehicle(s) in the forward path.
- Forward Collision Warning (FCW). An active safety feature that warns drivers in the event of an imminent frontal collision and provides a limited level of active braking to help slow down the vehicle and mitigate the potential forward collision.
- Lane Keeping Assistant (LKA). Prevents unintentional lane departures by detecting the presence of lines and warning the driver in the form of a visual warning and/or a torque applied to the steering wheel, and keeps the vehicle into the lane if requested.
- Active Blind Spot. Prevents unintentional collision with side moving objects by detecting the presence of moving objects in blind spots and warning the driver.
- Automated Highway Driving Assist (HAS). The system automatically adjusts steering angle, driving torque to maintain a road position and reduce fatigue for the driver.
- Traffic Sign Recognition. It informs the driver about the unconditioned speed limit and the possible presence of conditioned speed limits and “no overtaking ban”.
- Front and Rear Parking Sensors. Provides visual and audible indications of the distance between the front and/or rear bumper and a detected obstacle when backing up or moving forward, e.g. during a parking maneuvers.
- Surround View Camera. The system uses four cameras to monitor the area around the vehicle, placed on the front grid, under the side rear-view mirrors and on the trunk lid, between the number plate lights.

2.2.2 Sensing components

Each vehicle includes a set of sensor devices to obtain information about the position, speed and typology of the objects that surround the vehicle. In addition to this data, the vehicle will also provide information from the vehicle CAN network, including speed, acceleration, and collision and emergency break information. Concretely, the sensors installed on each vehicle are:





- Six surrounding high definition cameras, including four cameras with 120° of Field Of View (FOV), and two cameras with 60° of FOV.
- One Light Detection and Ranging (LiDAR) to identify objects around the car.
- Three radars (already in-vehicle) to detect objects around the car.

- The vehicles parking sensors incorporated on the front and rear of the vehicle to detect obstacles in the proximities of the car.
- A Global Navigation Satellite System (GNSS) to increase the accuracy, redundancy and the availability of the vehicle position, compared to the one already provided by the vehicle.

Sensing components on board Maserati Quattroporte (MY18)

In the following Table 8 are presented the components purchased and installed in the first vehicle:





Table 8: Components installed in the prototype 1

Component	Technical features	Units	Picture
Cameras	Sekonix SF3325-10X (60°fov)	2	
	Sekonix SF3324-10X (120°fov)	4	
GPS/GNSS	Xsens MTi-G-710-GNSSINS-2A8G4-DK	1	
LiDAR	Velodyne Lidar VLP-16 channels (PUCK)	1	

Sensing components on board Maserati Levante (MY19)

In the following Table 9 are presented the components purchased and installed in the second vehicle:

Table 9. Components installed in the prototype 2





Component	Technical features	Units	Picture
Cameras	Leopard Imaging LI-AR0231-GMSL (60°fov)	2	
	Leopard Imaging LI-AR0231-GMSL (120°fov)	4	
GPS/GNSS	Xsens MTi-G-710-GNSSINS-2A8G4-DK	1	
LiDAR	OUSTER OS1-64 channels	1	

For the Levante, both cameras and LiDAR were bought from a different supplier in order to carry out a benchmarking activity.

Sensing components on board Maserati Quattroporte (MY19)

In the following Table 10 are presented the components that will be installed in the third vehicle:

Table 10: Components installed in the prototype 3

Component	Technical features	Units	Picture
Cameras	Sekonix SF3325-10X (60°)	2	
	Sekonix SF3324-10X (120°)	4	
GPS/GNSS	Xsens MTi-G-710-GNSSINS-2A8G4-DK	1	
LiDAR	OUSTER OS2-128	1	




2.2.3 Communication components

4G-LTE antenna receivers have been installed to transmit information captured and processed by the vehicles. The switch to 5G receivers will be done as soon as the 5G technology is available in the market and it meets the requirements of the CLASS project.

2.2.4 Computation components

Each vehicle includes a powerful embedded high performance computing platform capable of implementing real-time data analytics. Those platforms are designed to process data from camera, radar, and LiDAR sensors to perceive the surrounding environment. Prototype 1 is based on a NVIDIA DRIVE PX2 Autochauffeur [16] (dual TX2 SOC plus 2 discrete Pascal GPUs). The NVIDIA DRIVE Xavier has been installed on Prototype 2, Maserati Levante, which is an ultimate version of the PX2 platform. The third vehicle, instead, will be equipped with the NVIDIA DRIVE AGX Pegasus.

Table 11: In-vehicle computing platforms

Vehicle	High-performance computing platform	Units	Picture
Quattroporte MY18	Nvidia DRIVE PX2 Autochauffeur	2	
Levante MY19	Nvidia DRIVE AGX Xavier	1	
Quattroporte MY19	Nvidia DRIVE AGX Pegasus	1	

2.2.5 Position and Field of View of the Sensors

Figure 28 summarizes the components installed in the car, as well as its position and the field of view of each one.

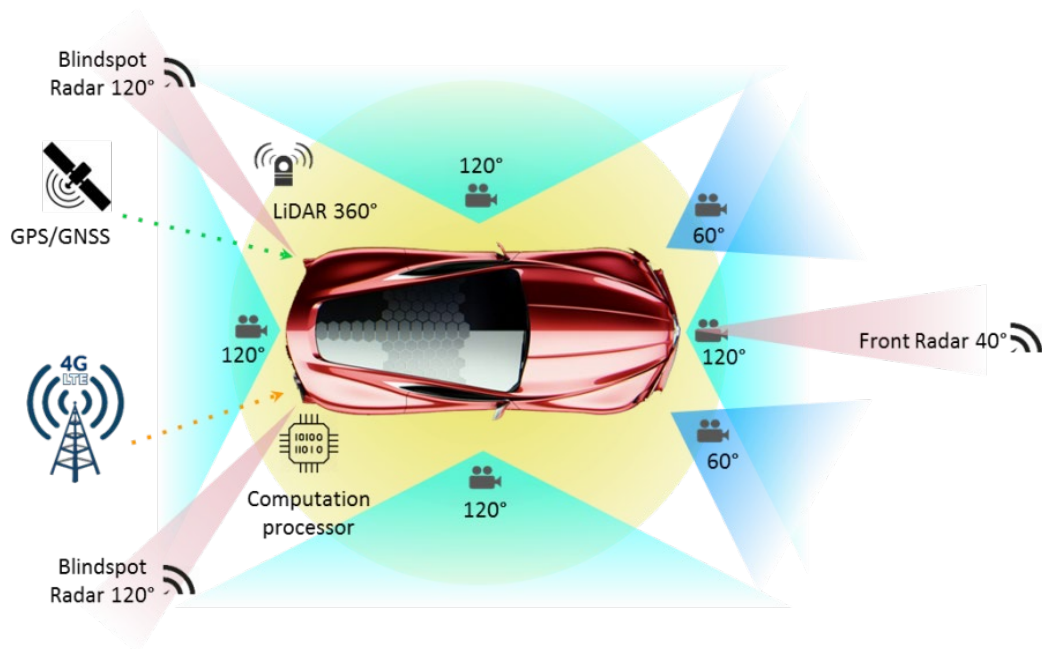


Figure 28: In-vehicle sensors (position and field of view)

All these components, integrated together and controlled by the computing platform, will allow the car to:

- Have a complete perception of its surroundings (thanks to the camera, LiDAR and radar).
- Position the car with respect to the maps (due to the GPS/GNSS receiver).

Exchange information with the infrastructure/fog node (thanks to the V2I communication) in order to extend, still more, the range of perception of the surroundings.

2.2.6 Vehicle's Software Architecture

As previously mentioned, the NVIDIA board is the control unit for the sensors installed, and the interface between the CLASS platform and the CAN bus network of the car. Figure 29 shows an overview of the vehicle's software architecture in CLASS.

Hereafter are listed the main activities carried out for the integration of the sensors (at hardware and software levels) in the car:

- **LiDAR**: installation of the LiDAR and acquisition of data.
- **Mapping** (with the support of the LiDAR): placement of the objects recognized and classification within the maps.
- **Cameras**: installation of all the 6 cameras, build the 360° field of view and acquisition of data.
- **Sensor/data fusion** between LiDAR and cameras.

- **V2I communication:** transmission of information from vehicle to the infrastructure/Fog node and vice versa.

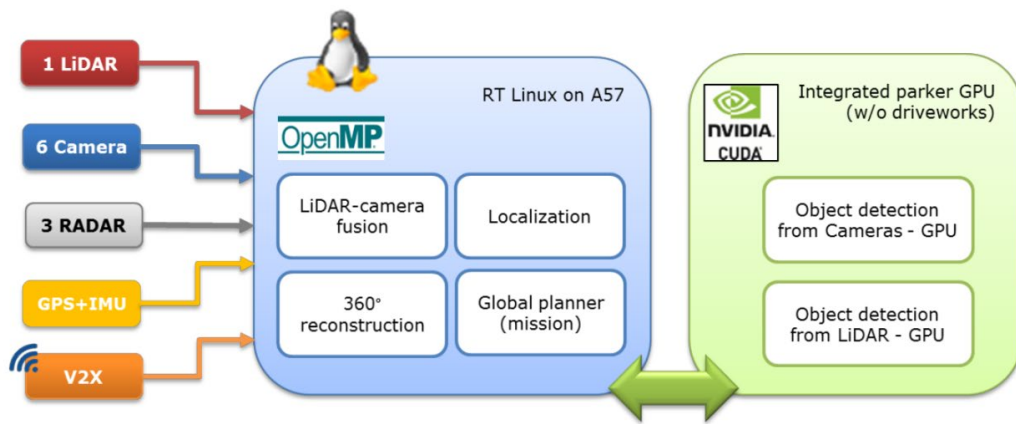


Figure 29: Vehicle's software architecture

3 Acronyms and Abbreviations

Each term should be bulleted with a definition.

Below is an initial list that should be adapted to the given deliverable.

- BB - Bounding Box
- CA – Consortium Agreement
- CAN – Controller Area Network
- CNN - Convolutional Neural Networks
- D – Deliverable
- DBSCAN - Density-Based Spatial Clustering of Applications with Noise
- DAG – Direct Acyclic Graph
- DBK - Data Knowledge Base
- DoA – Description of Action (Annex 1 of the Grant Agreement)
- EB – Executive Board
- EKF - Extended Kalman Filters
- EC – European Commission
- FOV – Field of view
- GA – General Assembly / Grant Agreement
- GPS – Global Positioning System
- HPC – High Performance Computing
- IPR – Intellectual Property Right
- KPI – Key Performance Indicator
- M – Month
- MS – Milestones
- MY – Model year
- PM – Person month / Project manager
- RADAR – Radio Detection And Ranging
- UC – Use Case
- WA – Warning Area
- WP – Work Package
- WPL – Work Package Leader

4 References

- [1] CLASS, "D2.6 - Second release of the CLASS software architecture," July 2020.
- [2] W. A. Kay, H. Andreas and K. Nagel, The multi-agent transport simulation MATSim, Ubiquity Press, 2016.
- [3] CLASS Deliverables, "D1.2 First release of the smart city use cases," 2019.
- [4] M. Verucchi, L. Bartoli, F. Bagni, F. Gatti, P. Burgio and M. Bertogn, "Real-Time clustering and LiDAR-camera fusion on embedded platforms for self-driving cars," in *IEEE Robotic Computing proceedings*, 2020.
- [5] "NVIDIA cuDNN - NVIDIA Developer," [Online]. Available: <https://developer.nvidia.com/cudnn>.
- [6] "Tensor Flow," [Online]. Available: <https://www.tensorflow.org/>.
- [7] "pywren," [Online]. Available: <http://pywren.io/>.
- [8] CLASS, "D5.4 - Final release of an augmented platform for analytics workloads," July 2020.
- [9] "Apache OpenWhisk," [Online]. Available: <https://openwhisk.apache.org/>.
- [10] J. Martí, A. Queralt, D. Gasull, A. Barceló, J. J. Costa and T. Cortes, "Dataclay: a Distributed Data Store for Effective Inter-Player Data Sharing," *Journal of Systems and Software*, vol. 131, pp. 129-145, 2017.
- [11] M. Baek, D. Jeong, D. Choi and S. Lee, "Vehicle Trajectory Prediction and Collision Warning via Fusion of Multisensors and Wireless Vehicular Communications," *Sensors (Basel)*, vol. 20(1), p. 288, 2020.
- [12] M. Alessio, G. Cristian, C. Andrea, C. Nicola and B. Paolo, "Graphic Interfaces in ADAS: from requirements to implementation," in *GOODTECHS*, 2020.
- [13] [Online]. Available: <https://github.com/apache/spark/tree/master/python>.
- [14] "Folium," [Online]. Available: <https://github.com/python-visualization/folium>.
- [15] E. Q. (. Luca Chiantore (MOD), "D1.1 – Use case Requirement Specification and Definition and First Description of the Sensing and Data-Sets Collected," 2018.
- [16] "NVIDIA DRIVE PX 2," [Online]. Available: https://docs.nvidia.com/drive/nvlib_docs/index.html .