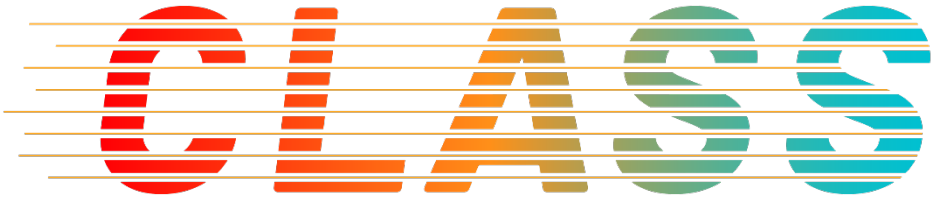

D1.6– Use case evaluation

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M42, 30th June 2021
Dissemination Level	PU
Nature	Report
Author(s)	Elli Kartsakli (BSC)
Contributor(s)	Luca Chiantore (MOD), Roberto Cavicchioli (UNIMORE), Eudald Sabaté (BSC), Vicente Masip (BSC), Eduardo Quiñones (BSC)
Reviewer(s)	
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 programme under grant agreement No "780622".*

Change Log

Version	Author	Description of Change
0.1	Elli Kartsakli (BSC)	First version
0.2	Luca Chiantore (MOD), Roberto Cavicchioli (UNIMORE),	Contributions on Section 2, Section 1.2.11 and Section 3.3
0.3	Eudald Sabaté (BSC), Vicente Masip (BSC)	Contributions on Section 3.1 and 3.2
0.4	Eduardo Quiñones (BSC)	Contributions on Section 3
1.0	BSC	Final version, ready to EC review.

Table of contents

Executive Summary

This deliverable reports the evaluation of the smart city use cases developed in CLASS. Three use cases have been selected to showcase the capabilities of the CLASS Software Architecture (SA): a real-time use case for the timely detection of potential collisions between road users, an air pollution estimation use case for the calculation of vehicle-polluted emissions, based on real-time traffic data, and an offline simulation framework for traffic management in smart city environments.

The deliverable is organized in three sections. First, an overview of the final release of the CLASS data analytics methods is given, providing links to the corresponding deliverables where the methods are fully described. Then, a description of the final infrastructure available at the Modena Automotive Smart Area (MASA) and the City of Modena data center is provided. Finally, the performance of the two real-time use cases (i.e., the collision detection and the air pollution estimation) is presented.

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

CLASS has developed a software architecture (SA) and a big-data analytics workflow to implement three use cases in the context of smart cities, evaluated in the living lab environment at the Modena Automotive Smart Area (MASA).

The main focus of CLASS has been laid on the implementation of two real-time use cases, as well as the development of a simulation framework for traffic management in smart city environments. The three CLASS use cases are the following:

1. The *collision detection application* focuses on the real-time detection of potential collisions and the generation of warning messages to alert the involved vehicles. The analytics are applied on data collected from the city camera infrastructure available at MASA, as well as from smart and connected cars, if available, and are executed across the compute continuum, from edge to cloud.
2. The *air pollution estimation application* estimates the vehicle-related pollution emissions in the MASA area, based on real-time information on the vehicles position, speed and type obtained by the CLASS data analytics. The output obtained estimates vehicle fuel consumption and emissions of NOx, PM, CO, HC and NO[1] at a time resolution of 1Hz, for each vehicle and road segment.
3. The *digital traffic signs (green routes) application* is a simulation framework for traffic management in smart city environments. This use case is not achievable in a real environment due to current traffic regulation. Our experimentation uses an urban transportation simulator, MATSim, with an extension introduced by CLASS. Specifically, the MASA area has been reproduced within the MATSim multi-agent simulator and four additional modules have been added to the MATSim simulator for enhancing its ability to simulate smart city related scenarios. These additions consist of a *communication module* and a *smart agent module*, in order to allow agents to send data to the software architecture, a *perception module*, in order to allow cameras to locate not equipped agents, and finally *an analysis module* to register values needed for the analysis.

This section describes the final version of the *combined big-data analytics workflow* for the implementation of the aforementioned CLASS use cases.

1.1 Big-data analytics methods overview

Figure 1 shows the combined big-data analytics workflow as a Direct Acyclic Graph (DAG) in which nodes represent the different analytics methods (the number in the node identifies the method) and edges represent the data exchange (and so dependencies) among the different analytics.

[1] NOx: Nitrogen Oxides; CO: Carbon monoxide; PM: Particulate Matter; HC: Hydrocarbons

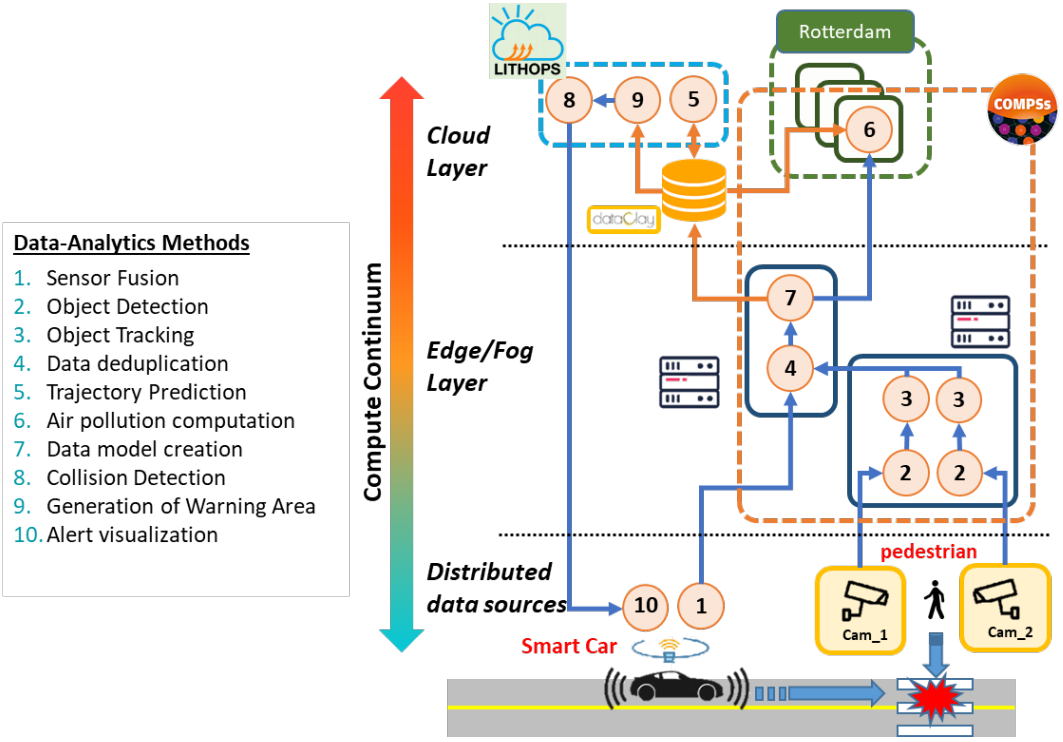

Figure 1. The combined big-data analytics workflow

The depicted analytics methods are summarized as follows.

1. **Sensor fusion**. This method is responsible for the detection and tracking of objects, combining the data collected by the different sensors available in the smart and connected vehicles.

2. **Object detection**. This method is responsible for the detection of objects and the computation of their position based on raw data from the city cameras in the MASA.

3. **Object tracking**. This method tracks the movement of objects detected and identified with *method 2*, across multiple video frames.

4. **Data deduplication**. This method identifies and removes multiple detections of the same object, when simultaneously captured by multiple cameras of the city.

5. **Trajectory prediction (TP)**. This method predicts the trajectory of road users (cars, bicycles, pedestrians, etc.) based on their history of tracked positions provided by *method 3.*

6. **Pollution computation**. This method estimates emissions from the drive cycles of vehicles identified by *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 storing the information of the data analytics (methods 1-6) into a common data model and aggregating it to the *Data Knowledge Base (DKB)*.

8. **Collision detection (CD)**. 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 generates a Warning Area with respect to a connected vehicle. Only objects in the WA of the vehicle are then considered as relevant to for the calculation of *method 8*.

10. **Alert visualization**. This method, executed on connected vehicles, allows the visualization of potential collisions between the connected vehicle and other road users within its WA.

11. **Predictive models**. This method implements the digital traffic signs application on the MatSim simulator [1].

1.2 Implementation of big-data analytics methods

The big-data analytics methods have been first described in D1.2 [2], and then refined in D1.4 [3]. This section will provide a brief overview for convenience, and report any modifications and updates since the last reported version.

1.2.1 Sensor fusion

The sensor fusion method, executed in fully sensorized smart cars, aims to fuse the data from cameras and LiDARs (Light Detection and Ranging) in a computationally efficient way, so as to meet the real-time requirements of the application. Detailed updates to the sensor fusion task can be found in deliverable D3.6 [4]. The main change with respect to what previously reported is the update of the Deep Neural Network (DNN) version for object detection, which has been upgraded from Yolo v3 to Yolo v4 to increase precision while keeping almost the same computational performance.

1.2.2 Object detection

The object detection method is applied on the video streams obtained by the street cameras in the MASA. The main concept behind the object detection in CLASS is the following. First, objects are detected and classified with an optimized version of Yolo that runs on the tkDNN framework. Then, the global position of each detected object is computed. 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. Detailed updates to the object detection task can be found in deliverable D3.6 [5].

1.2.3 Object tracking

The aim of the tracking method is to follow the detected objects, tracking their movement within the area covered by each camera. Each tracker is associated with a video source and assigns a unique id to each tracked object. Detailed information on the object tracker task can be found in deliverable D3.6 [5].

1.2.4 Data deduplication

The aim of the data deduplication method is to identify and manage the road users that are detected by different actors (i.e., cameras or smart cars) which partially cover the same area. More details on the data deduplication task can be found in deliverable D3.6 [5].

A more detailed insight about how object IDs are handled by the deduplicator in order to allow the trajectory prediction and collision detection to address correctly the connected vehicles requests is the following. The data deduplicator considers as key the pair of camera ID and tracking ID, since each camera have its own tracking ID generated by the tracker. When two road users of the same category are within a defined threshold, the deduplicator keeps a vector of keys for each road user to maintain history of the previous IDs. In this way we are able to assign to each road user the same key and predict correctly its path when passing information to the trajectory predictor.

1.2.5 Trajectory prediction (TP)

The aim of the trajectory prediction is to provide an estimation of the most likely future positions of a road user (vehicle or pedestrian) based on the history of their tracked positions. The enhancements that have been introduced with respect to the work reported in D1.4 [3] will be described in continuation.

In the first place, the implementation of the TP algorithm has been improved with the following modifications:

- The number of predicted points and the time interval between consecutive predictions has been added as a configurable parameter. In the final evaluation, the estimation of future 8 points with an interval of 500 ms has been selected, corresponding to an overall trajectory prediction of 4 seconds.
- Two different implementations of the regression algorithm have been added, that enable the execution of regressions with a configurable degree (e.g., 1st degree linear regression, 2nd degree quadratic, etc.).
    - The first regression method is actually an extension of the one reported in D1.4 [3], in which the trajectory past points are fit to a polynomial equation using the least-squares method.
    - The second regression method has been implemented by using the Scikit-Learn[2] open-source framework. In this method, the tool's functions (concretely PolinomialFeatures[3]) are used to fit the trajectory data into a polynomial of variable degrees. More specifically, by specifying the maximum number of degrees for this polynomial, the tool returns a matrix with all the combinations of features with a degree less or equal to the one specified. Once this matrix is obtained, it is fed into a Linear Regression

[2] https://scikit-learn.org/stable/
[3] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

model (LinearRegression[4]) that fits to the values of the matrix and then, predicts the new trajectory values. This method can reduce the complexity (number of degrees) of the polynomial if it receives as input a high number, which may produce overfitting, but has a similar performance to the first method for low regression orders (i.e., first and second degree).

In the final evaluation, regressions of first and second order have been employed, to better match the road segment covered by the different cameras (i.e., whether the covered area focuses on a straight road segment or a curve).

In the second place, two dictionaries have been added as input to the TP method, determining the minimum and maximum amount of historical positions per type of object (i.e., car, pedestrian, etc.) to be considered for the regression-based prediction method. Higher values for the minimum number of considered points usually results to more stable predictions for the trajectory but have the drawback of requiring the object to be tracked for a longer time (which makes the algorithm more susceptible to lost frames or tracking errors).

```
# min amount of trajectory points to start predicting per class
QUAD_REG_MIN_DICT = {
    "person":20,
    "car":10,
    "truck":10,
    "bus":10,
    "motor":10,
    "bike":10,
    "rider":20,
    "train":10
}
```

Figure 2. Minimal historical points for predictions

```
# max amount of trajectory points to manage per class
QUAD_REG_LEN_DICT = {
    "person":50,
    "car":40,
    "truck":40,
    "bus":40,
    "motor":40,
    "bike":40,
    "rider":50,
    "train":40
}
```

Figure 3. Maximum historical points for predictions

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

The values selected for the two dictionaries in the final evaluation are shown in Figure 2 and Figure 3, and have been set after an evaluation phase. For this evaluation, a comparison between the actual and predicted trajectories of road users in a recorded video have been made, considering different combinations of minimum historical points and number of predicted positions. An implementation of the Haversine[5] formula has been used to calculate the shortest distance in meters between the predicted and the real positions of each object, and the Root-Mean-Square Error (RMSE) of the distances has been used as the evaluation metric.

An example of the evaluation process for two types of road users (pedestrian and cars) is shown in Figure 4, applied on a recorded video from the camera 20939 at the MASA. The output shows the RMSE values for different combinations of historical points (5, 10, 20, 30, 40, 50) and predicted points (1, 5, 10, 20, 30). The selected combinations for each vehicle have been the ones that yield a relatively RMSE value with the smallest possible number of historical points. For example, for a prediction of 5 points ahead in time for pedestrians, a sufficiently small error can be obtained with 20 historical points, whereas for cars, the reasonably good performance can be achieved with only 10 historical points.

```
ID: 20939_1
TYPE: person
RMSE (historical points, predicted points): RMSE value

RMSE (5,1): 1.301734911867662
RMSE (5,5): 5.362693396604704
RMSE (5,10): 15.738824410468842
RMSE (5,20): 55.047539130027225
RMSE (5,30): 131.32912349887755

RMSE (10,1): 1.282678118322195
RMSE (10,5): 4.023164294390557
RMSE (10,10): 9.716558414282762
RMSE (10,20): 31.287757732390933
RMSE (10,30): 68.86208820756251

RMSE (20,1): 1.515372105321413
RMSE (20,5): 3.1466499366585365
RMSE (20,10): 6.318899982450494
RMSE (20,20): 17.35207894884042
RMSE (20,30): 35.254050659823704

RMSE (30,1): 1.4481138586794393
RMSE (30,5): 2.1572421872167302
RMSE (30,10): 3.8291895459685956
RMSE (30,20): 6.157003663196969
RMSE (30,30) not enough data

RMSE (40,1): 1.5119852787300498
RMSE (40,5): 2.3001835225260567
RMSE (40,10): 3.1009234168049526
RMSE (40,20) not enough data
RMSE (40,30) not enough data

RMSE (50,1): 1.6400877040716921
RMSE (50,5) not enough data
RMSE (50,10) not enough data
RMSE (50,20) not enough data
RMSE (50,30) not enough data
```

```
ID: 20939_27
TYPE: car
RMSE (historical points, predicted points): RMSE value

RMSE (5,1): 0.6000739317004512
RMSE (5,5): 2.321078762349682
RMSE (5,10): 6.403833908555203
RMSE (5,20): 23.25511506192327
RMSE (5,30): 53.09223741613018

RMSE (10,1): 0.44017165018108345
RMSE (10,5): 0.946498512863295
RMSE (10,10): 2.155286133353007
RMSE (10,20): 6.6639620260484325
RMSE (10,30): 14.76516327067647

RMSE (20,1): 0.3625916345968366
RMSE (20,5): 0.5628098298007016
RMSE (20,10): 0.9153096561097027
RMSE (20,20): 2.161846793640586
RMSE (20,30): 4.337331514134081

RMSE (30,1): 0.36799799290572033
RMSE (30,5): 0.482643784608555
RMSE (30,10): 0.7347142516145153
RMSE (30,20): 1.6648609960771255
RMSE (30,30): 3.014112485890962

RMSE (40,1): 0.3987633676586011
RMSE (40,5): 0.5540689512742931
RMSE (40,10): 0.8104548058640324
RMSE (40,20): 1.4605059104697617
RMSE (40,30): 2.4620445547304857

RMSE (50,1): 0.43575993345120967
RMSE (50,5): 0.5340360090563977
RMSE (50,10): 0.7123938713916848
RMSE (50,20): 1.1208474957536865
RMSE (50,30): 1.8708494495351007
```

Figure 4. Evaluation of two trajectories for two types of road users (pedestrian and car)

In CLASS, the TP is executed over the data available in the DKB in the following way. A dataClay filtering method has been implemented, that retrieves from the DKB all

[5] https://www.movable-type.co.uk/scripts/latlong.html

relevant objects and their events on which TP must be applied, with the following properties:

- it returns only those objects with sufficient number of recently tracked positions (i.e., at least equal to the minimum number of historical position as defined in the dictionary of Figure 2).
- it returns a maximum of recent events, as determined by the dictionary of maximum historical positions of Figure 3.
- It ensures that only objects with recently updated events are considered

A Lithops wrapper function retrieves all the relevant information from dataClay, divides them into chunks of a configurable size (three by default) and triggers the Lithops actions for the concurrent estimation of predicted trajectories for each chunk based on the selected regression method. The predicted trajectories are then appended to the corresponding objects in the DKB.

1.2.6 Data aggregation

After the execution of the object detection, tracking and deduplicator methods, the extracted knowledge on the road users at the MASA is stored in a data model and aggregated into the DKB. This information is then updated by the trajectory prediction method, which adds prediction estimations for all objects with sufficient recent historical positions.

Ultimately, all CLASS use cases are built upon the information aggregated in the DKB: the collision detection method is applied on all the vehicles with valid trajectory predictions, whereas the air pollution estimation employs information on the type and speed of the detected vehicles.

The information included in the DKB is structured in the data model presented in Figure 5 and implemented with dataClay [6]. The final Python implementation of the model, with some changes with respect to D1.4 [3], includes the following classes:

- *DKB*: it corresponds to the main Python class and contains the following information:
    - Kb: a dictionary of EventsSnapshots, maintained for a given (configurable) period of time (e.g., 15 minutes)
    - K: the maximum number of the most recent *Events* to be returned to the trajectory prediction and collision detection methods. The rest of Events are considered aged and not relevant for these calculations.
    - connectedCars: a list of the ids of the connected cars
    - objects: dictionary with object string identifies, used internally for communication with the TP method
    - federatedObjects: detected objects which have been federated to the DKB
- *EventsSnapshot*: It contains a list of events (*events_list*) obtained by the analytics at a given *timestamp*.
- *Event*: the event represents the information regarding the position and movement of a detected *Object* at a given *timestamp*. Each event consists of:

- id_event: a unique identifier of the event
- timestamp: the timestamp at which the detection of the object took place
- longitude_pos, latitude_pos: the GPS coordinates of detected objects
- detectedObject: points to the Object with which the event is associated
- speed and yaw: the instantaneous speed and orientation of the object
- geohash: the geohash code corresponding to the GPS position of the object (see section 1.2.8 for more information)

- Object: it represents the objects detected by the analytics. Each object consists of:
  - id_object: a unique identified of each object, assigned by the tracker (Section 1.2.3)
  - type: the type of the detected object (i.e., person, car, truck, bus, motorcyle, bike, train)
  - events_history: a dictionary with the Events associated with the object
  - trajectory_px, trajectory_py, trajectory_pt: it contains the most recent output of the trajectory prediction method for the object, i.e., a list of the GPS coordinates for each predicted point for a given time in the future
  - retrieval_id: string identifier used internally for object retrieval by the TP method

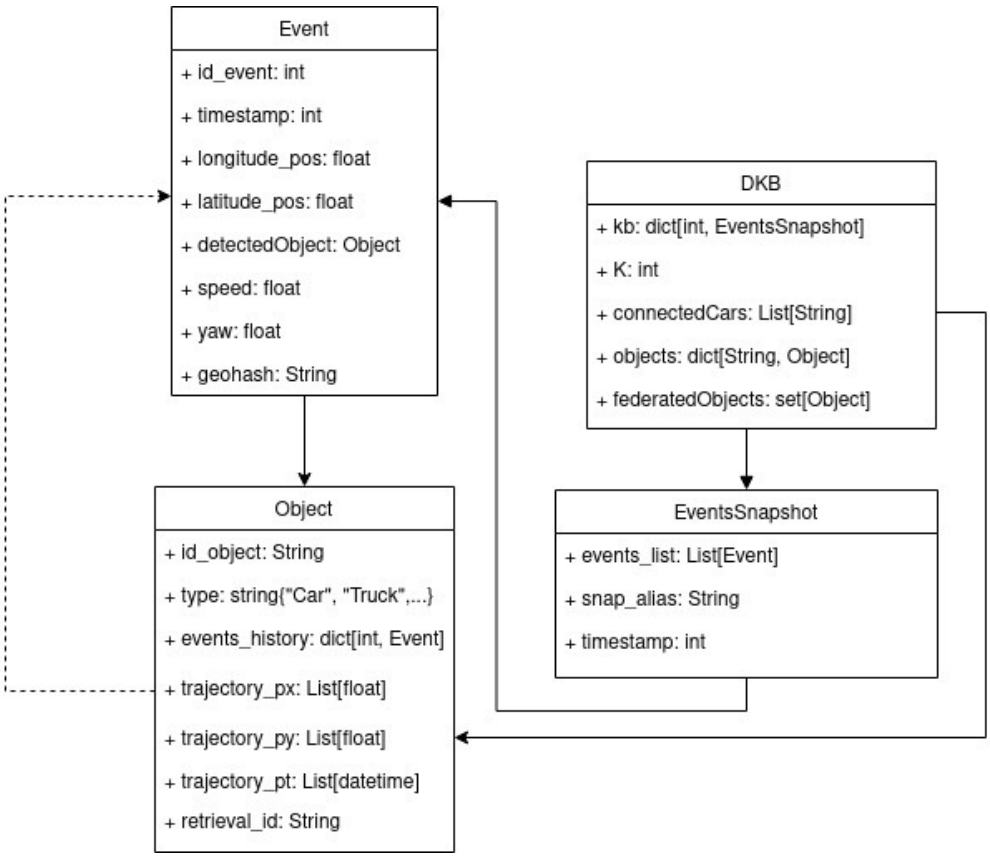

Figure 5. Diagram of the data model for the information contained in the DKB

Furthermore, a method for the periodic cleaning of the DKB has been implemented. The main idea is that EventSnapshots older than a given (configurable) age threshold are eliminated

from the data model. Once Events are deleted, any Object that remains with no associated events is also removed.

1.2.7 Collision detection (CD)

In the previous implementation of collision detection, reported in D1.4 [3], a collision detection was triggered when the trajectories of two road users intersected at the same future point in time. Since then, a substantial update of this method has been provided, defining a surrounding area[6] around objects to account for the volume of the vehicles and the unpredictability of pedestrian movements.

Specifically, an elliptical area with a width of 1.5 meters for each side of the vehicle has been defined for vehicles, whereas a triangular area of 50 degrees centered along the predicted linear trajectory has been considered for pedestrians. This modification has led to a more accurate detection of collisions, since the representation of objects is more realistic.

An example is shown in Figure 6, with the current implementation of the CD depicted in the right part of the figure, and the previous implementation reported in D1.4 [3] shown at the left. For both plots, the TP calculation is exactly the same, with the blue line corresponding to the predicted trajectory of a car, and the green line corresponding to a pedestrian. In the previous implementation (left), the collision is defined as the intersection the two predicted trajectory lines, which can be interpreted as the potential event of the two objects being at the exact same position in a future moment in time. In the current implementation (right), collision is detected when there is an overlap in the surrounding areas of each type of vehicle, drawn around the predicted trajectory points (corresponding to the same moment in time. It is evident that the current approach can capture more realistic scenarios, e.g., when the collision of a pedestrian with the lateral part of a vehicle. Furthermore, detections may be detected earlier in time, as shown in this example, since the overlapping of the surrounding areas occurs before the intersection of the two trajectories.

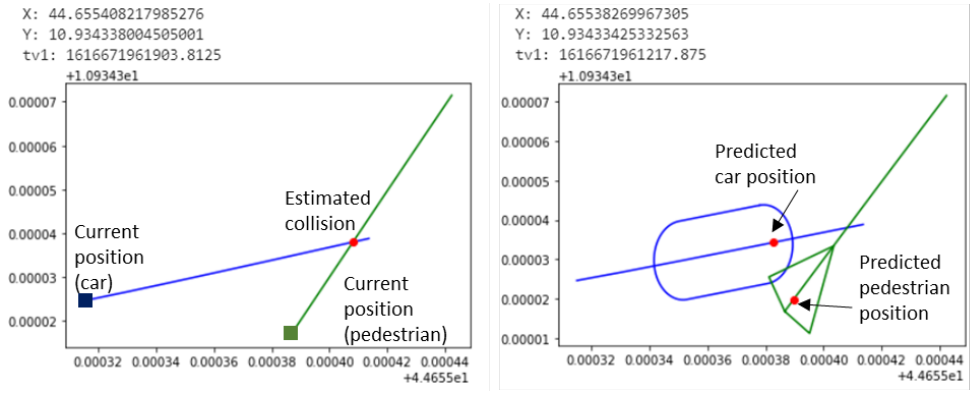

Figure 6. Example of collision detection implementation. Left: Initial implementation reported in D1.4, with collision detected as an intersection of predicted trajectories.

[6] The Python library Shapely has been used to that end: https://pypi.org/project/Shapely/

Right: Final implementation, with collisions detected as an overlapping of surrounding areas introduced around vehicles (elliptical) and pedestrians (triangular)

Furthermore, the current method can capture potential collisions for objects that are close to each other, even when their trajectories are almost parallel, as shown in the example of Figure 7.

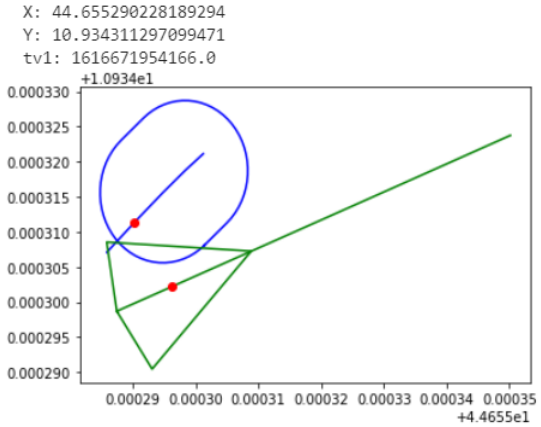

Figure 7. Collision detection in close trajectories

In CLASS, the CD is invoked asynchronously over the data available in the DKB in the following way. A dataClay filtering method has been implemented, that retrieves from the DKB all relevant objects with predicted trajectories. Then, the CD can be applied either to all the detected moving objects, checking the possibility of collision between all pairs of objects, or to only a subset of reference vehicles (e.g., connected cars) that request the computation of CD, depending on the desired scenario. In the second case, the Warning Area (WA) of the reference vehicle is first calculated and potential collisions between the reference vehicle and all the objects within the WA are then obtained.

1.2.8 Warning Area generation

The geohash[7] encoding has been selected for the representation of the warning area around a vehicle. The geohash is a geocode system which encodes a geographic location into a short alphanumeric string. A geohash identifies a rectangular area in the map, whereas the number of digits defines the area of the cell.

In CLASS, a precision of 7 digits has been selected, partitioning the map into square areas of 153m x 153m. When the CD is invoked for a specific object (e.g., a smart/connected car), the Warning Area is defined as the area within the same geohash, as well as the neighboring cells.

1.2.9 Alert visualization

The visualizer for the alert warnings is a 3D visualization tool written with the Godot game engine[8]. This engine has been selected to facilitate the portability of the visualizer on different devices, such as the edge platform in smart and connected cars,

[7] https://www.movable-type.co.uk/scripts/geohash.html
[8] https://godotengine.org/

as well as in the control room. The visualizer can receive data from the deduplicator to show all the road users detected by the workflow, or it can receive data from the collision detector to visualize the warning for the connected vehicle.

A snapshot of the 3D visualizer showing the generation of a warning alert for a potential collision between two vehicles in shown in Figure 8.

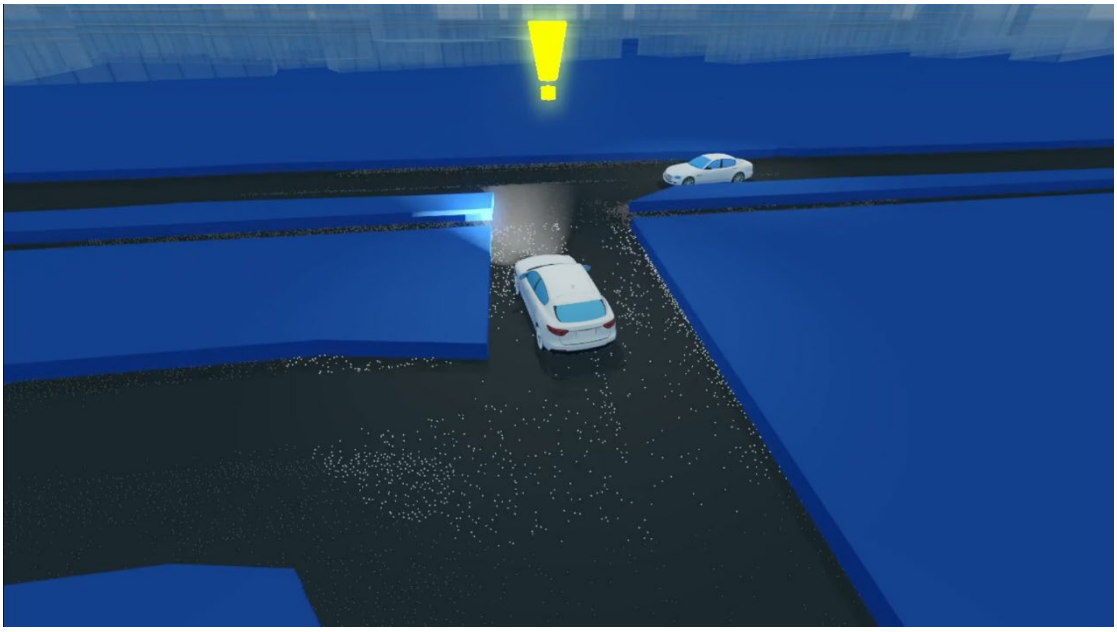

Figure 8. The 3D visualizer showing a warning for a potential collision between two vehicles

1.2.10 Air Pollution estimation (and visualization)

The air pollution estimation aims to infer vehicle-related pollution data based on the real-traffic information obtained by the street cameras in the MASA. This information is passed on the PHEMlight model [7], a micro-scale model for simulation of fuel consumption and instantaneous pollutant emissions of on-road vehicles. PHEMLight can distinguish between different types of cars (i.e., passenger cars, light, and heavy duty vehicles) and consider the corresponding emission classes (see D1.2 for detail).

The PHEMLight estimation method can be called periodically to obtain the vehicle-related information either at the fog layer (from the output of the deduplicator) or the cloud layer (at the DKB). Then the PHEMLight model is applied, generating the pollution emission estimations for the given time intervals.

A visualization dashboard has also been developed as a web application, in order to show the obtained air pollution estimations on an interactive map. The dashboard prints the average pollutant values on the map, which can be selected by a drop-down menu, associating portions of the area (i.e., road segments) to the area covered by each street camera in the MASA. A snapshot of the dashboard is given in Figure 9.

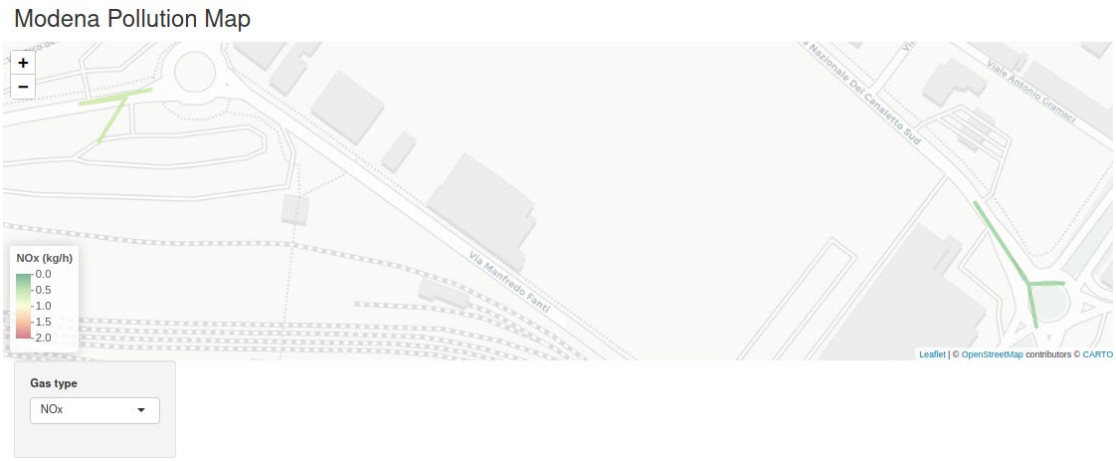

Figure 9. Visualization of the PHEMLight output for two camera inputs at the MASA

1.2.11 MATSIM

The baseline version of MATSim does not enable the interaction among vehicles and the city infrastructure. Moreover, vehicles' daily plans and corresponding routes are fixed at initialization time and do not change during the simulation runtime. Since this is a limitation that would prevent developers from testing emergency response mechanics, a MATSim extension has been used for allowing the developers to implement dynamic agent routing. Starting from the MATSim extension, CLASS has further developed the dynamic agent planning module so as to be able to dynamically adjust agents' routes according to messages exchanged within the city infrastructure. The city infrastructure is implemented as an extension of MATSim. It is able to listen to the simulation's events in order to obtain a global view of the city state. Position and communication capabilities of the involved vehicles are part of this observed state. In summary our developed extension allows the vehicles to communicate with the city infrastructure and thus, dynamically change their behaviour based on the information given by city infrastructure itself.

First, a representation of the MASA area street network was created. We used several plugins to import to MATSim the original map from the OpenStreetMap (OSM) database and then we augmented the network with additional information such as capacities for the road links, turning bans and parking spaces. The MASA area is a 1-square km wide area centered around the coordinates (44.65632, 10.93150) in OSM. The size of the simulated population was estimated using regional traffic flow data made available online; using such data we estimated a rush-hour road users population between 10000 and 20000 agents.

For each experimental scenario we provide an XML file as required as input by MATSim. In such a file, we describe the initial plans for the agents, i.e. their departure and arrival locations. Plan are initially set for all the interested agents related to all the different experimental scenarios so to that the agents plans remain the same while we simulate different behaviors. Agents' plans are constructed taking two random locations on the MASA map: these two places represent home and work locations. The agents perform two trips in a day, first from home to work, then from work to home.

The departure time from home and work is chosen following a normal distribution with peak on typical rush hours (8:00 AM–9:00 AM for home departure and 06:00 PM–07:00 PM for departing from work). With regard to traffic modelling, simulated vehicles' routes undergo a process of calibration for depicting realistic traffic situations. It is important to highlight, however, that the purpose of the test is not to provide a realistic reconstruction of the traffic in the MASA area. More specifically, we are interested in the *mechanisms* behind emergency response while varying the capabilities of the involved vehicles within a relatively restricted city area.

Therefore, we argue that once one-way streets, turning bans, street lane directions/capacities extracted from OSM and a reasonable distribution of peak/rush hours are set for the simulation, this constitutes a proper environment for studying our proposed emergency response strategies.

2 Description of the Modena Automotive Smart Area (MASA)

2.1.1 Cameras and smart cameras

Figure 10 shows the final position of the cameras (depicted as red markers) installed in the MASA, in Modena, Italy.

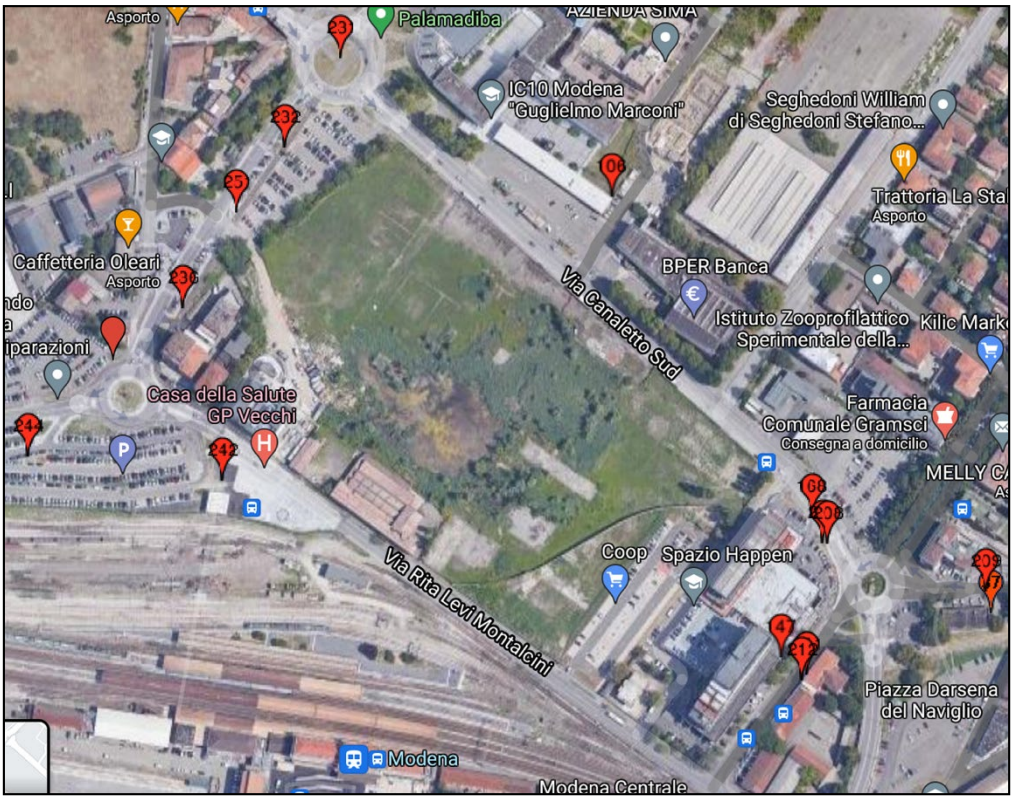

Figure 10. Map of the installed cameras in MASA

In detail, the cameras include:

- **1 four optic camera with 360º overview (number 231):** this device (model "*Axis Q6000-E Mk II*") has four 2MegaPixel sensors that provide a full detection and control of the roundabout (Figure 11). It also has computational capability on-board, but not programmable by the CLASS partners. For this reason, this device is connected to a specific edge-node hosted in the Modena data center, which manages the extraction of metadata from images by means of a trained DNN.

Figure 11. 360 angle view of the Axis Q6000-E Mk II model

- 14 traditional bullet cameras (numbers 106, 147, 168, 207, 208, 209, 211, 212, 232, 236, 242, 244, 251) of various suppliers. In particular the "*Axis P1367-E 5MP Outdoor Box Network Camera"* model (numbers 208, 209, 211, 212, 232, 236, 251) provides images optimized for detection or forensic purposes regardless of light conditions, with the following key features, shown in Figure 12.

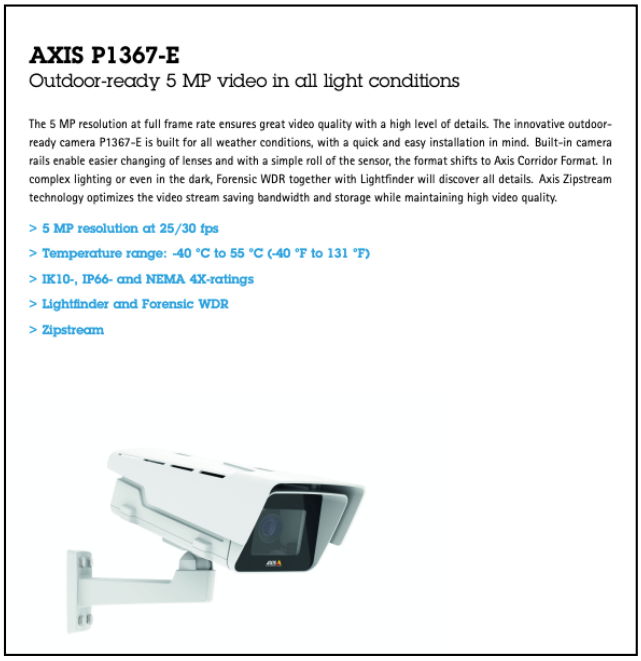

*Figure 12. Technical overview of street camera specifications, model "*Axis P1367-E 5MP Outdoor Box Network Camera"

Figure 13. Example of the quality of capture from the Axis P1367-E model

These devices have a high quality resolution but don't have any on-board computation resources. Hence, they are connected to four edge nodes that manage the data acquisition.

The installation and testing of the new smart cameras, named "Mind City", was not completed within the lifetime of CLASS, due to the impact of COVID which affected all the installation activities. However, the cameras are currently being installed, thus enhancing the MASA monitoring capability, even after the completion of CLASS. These cameras have two full HD sensors and on-board and a TX2 GPU. The design and some technical specifications of this camera are shown in Figure 14.

Figure 14. Technical overview of the camera model "Mind City Beta"

2.1.2 Other IoT devices

Some additional IoT devices installed in the MASA during the CLASS project are summarized next. These sensors were not exploited by the CLASS use case within the lifetime of the project, due to time limitations. However, their availability will be exploited in the future by the City of Modena, as part of the sustainability efforts to benefit from the outcome of the real-time CLASS use cases.

- 2 Pollution sensors ("Libelium Wasp mote – Smart environment" model), connected to the Lo-Ra network, for the detection of the following air quality parameters: carbon monoxide (CO), carbon dioxide (CO_2), nitrogen dioxide (NO_2); particulate matter (PM). The accuracy of the detection of these instruments is calibrated using as comparison the data of the official survey stations of the Regional Agency for the environment.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	Reference
A, B, C or F	Carbon Monoxide (CO) for high concentrations [Calibrated]	9371-P
	Carbon Monoxide (CO) for low concentrations [Calibrated]	9371-LC-P
	Carbon Dioxide (CO_2) [Calibrated]	9372-P
	Oxygen (O_2) [Calibrated]	9373-P
	Ozone (O_3) [Calibrated]	9374-P
	Nitric Oxide (NO) for low concentrations [Calibrated]	9375-LC-P
	Nitric Dioxide (NO_2) high accuracy [Calibrated]	9376-HA-P
	Sulfur Dioxide (SO_2) high accuracy [Calibrated]	9377-HA-P
	Ammonia (NH_3) for low concentrations [Calibrated]	9378-LC-P
	Ammonia (NH_3) for high concentrations [Calibrated]	9378-HC-P
	Methane (CH_4) and Combustible Gas [Calibrated]	9379-P
	Hydrogen (H_2) [Calibrated]	9380-P
	Hydrogen Sulfide (H_2S) [Calibrated]	9381-P
	Hydrogen Chloride (HCl) [Calibrated]	9382-P
	Hydrogen Cyanide (HCN) [Calibrated]	9383-P
	Phosphine (PH_3) [Calibrated]	9384-P
	Ethylene (ETO) [Calibrated]	9385-P
	Chlorine (Cl_2) [Calibrated]	9386-P
D	Particle Matter (PM1 / PM2.5 / PM10) - Dust	9387-P
E	Temperature, humidity and pressure	9370-P
	Luminosity (Luxes accuracy)	9325-P
	Ultrasound (distance measurement)	9246-P

Figure 15. Technical overview of pollution sensors

- 10 Lo-Ra parking sensors.

These devices, installed on the surface of parking spaces, allow the detection of free parking slots (the arrival and departure of vehicles), by means of electromagnetic field sensor.

Figure 16. Photo of a Lo-Ra parking sensor

- 6 traffic lights counters for passing vehicles, made by magnetic loops, represented by green lines in the figure below.

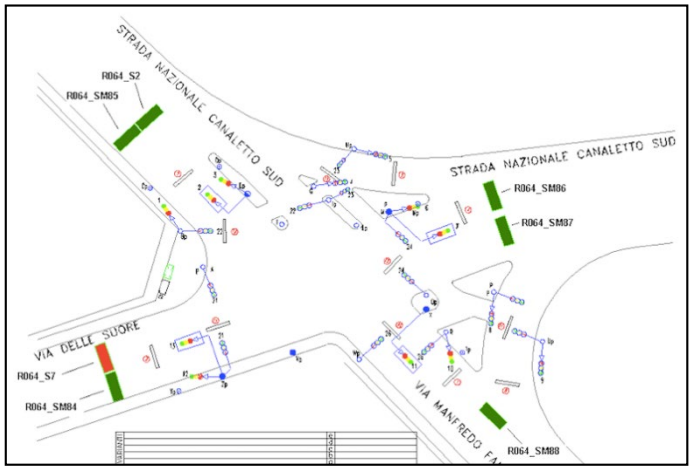

Figure 17. Map of positioning of traffic light counters

2.2 Smart and connected vehicles

Figure 18. The Maserati Levante (at the left with the CLASS logo) and another Maserati car made available for the execution of the use cases

CLASS has employed a Maserati Levante, a mid-size luxury crossover SUV (Sport Utility Vehicle) for the evaluation of the use case scenarios. 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. Additional vehicles, not exclusively used by CLASS, have also been used during the execution of the use cases, to simulate hazardous driving conditions that could trigger an alert. More details on the employed car and the equipped sensors can be found in D1.4 [3].

2.3 Modena communication infrastructure

MASA provides interconnected wired and wireless networks, detailed in continuation.

2.3.1 Wired network

A ring of optical fiber that interconnect the urban racetrack and the data center. The following map in Figure 19 shows the roads, roundabouts, bridge and parking areas that are interested by the project and that actually host devices and sensors.

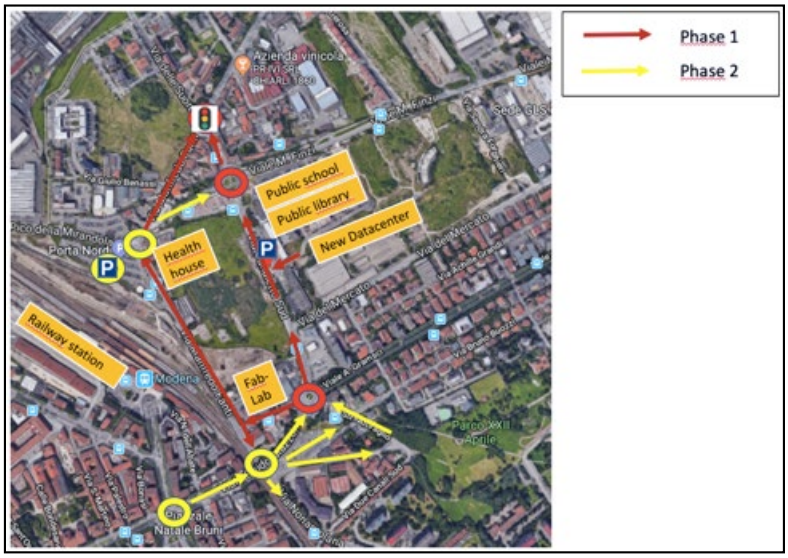

Figure 19. Optical fiber interconnections in the MASA

2.3.2 Wireless network

We implemented three different types of connectivity, private 4G (4.5G), 5G and Long Range (LoRa) network, with dedicated antennas, shown in Figure 20.

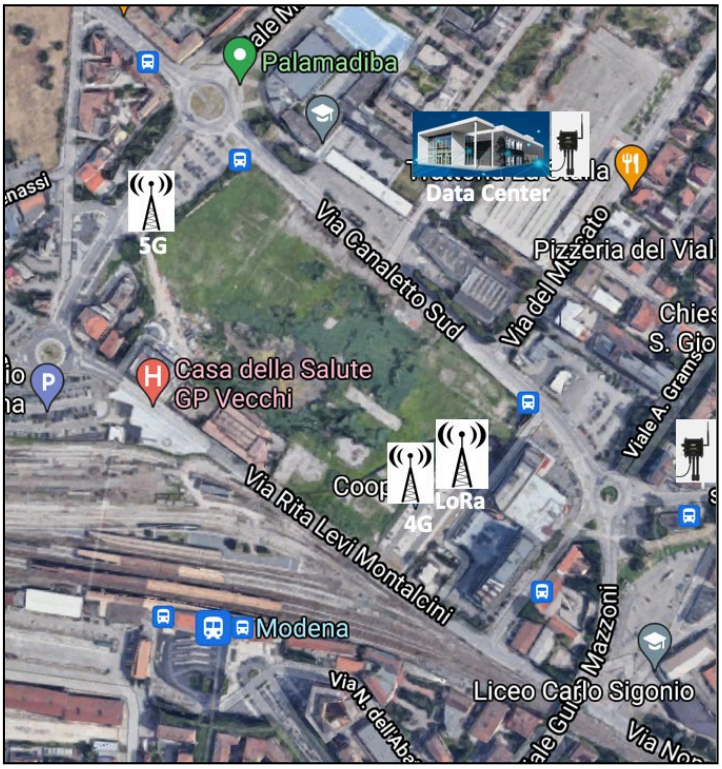

Figure 20. Map of private antennas available to CLASS

2.3.2.1 Private 4G (4.5G) network

The LTE solution acquired is defined 4.5G in terms of functionality. We installed on-site a 4G/4.5G dedicated infrastructure, both the antenna (installed on top of the highest building of the MASA, as shown in Figure 21), the radio component (eNB LTE) and the functionality of Core (Full EPC), implementing a private local area network.

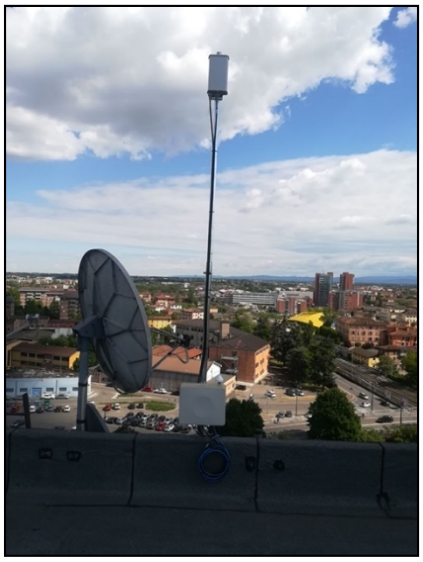

Figure 21. Photo of the installed antenna for the private cellular network used in CLASS

The configuration identified guaranteed service levels, traffic capacity, low latency and the possibility of Network Slicing approximate to those envisaged in 5G technology, not yet available on the market in the first half of the project. This technical solution received the award from the ICCA during the Critical Communication World Event in Berlin[9].

2.3.2.2 Long Range network (LoRa)

Long Range (LoRa) network is a network infrastructure to interconnect sensors and devices with low bandwidth needs, in our case parking and pollution sensors.

We have implemented the LoRa Wide Area Network (LoRaWAN) protocol that supports long range, low-cost, mobile, energy-efficient and secure end-to-end bi-directional communication for Internet of Things (IoT) and Machine to Machine (M2M) applications. It operates on license-free and cost-free Industrial, Scientific, Medical bands – EU 868 MHz.

[9] https://www.athonet.com/athonets-private-lte-volte-in-the-deepest-mine-in-the-americas/

2.4 Modena computing infrastructure

2.4.1 Edge Computing Infrastructure

The MASA provides four x86-64 servers employed as edge nodes with the following features:

Table 1. Edge nodes capabilities

Feature	Description
CPU	Intel® Xeon E3-1245 v.5
RAM	32 GB
Hard-Disk	Samsung SSD PM951 256 GB
LAN	Gigabit Ethernet
GPU	NVIDIA Volta gpu (TitanV)

Each edge node manages a cluster of traditional bullet cameras, giving them the computational capability, extracting metadata from images by means of trained neural network.

2.4.2 Cloud Computing Infrastructure

The City of Modena provides the computation and storage capabilities of its data center. In particular it provides a virtualization cluster infrastructure and a storage area network, with the following capabiltiies:

- *3 bare metal hosts with two CPU Intel Xeon® Gold 5120, 2.20Ghz (14 cores for sockt) and 785 GB of storage;*
- *1 Storage Area Network* (SAN) EMC2, model VNX-5400;
- *1 Virtualization platform* VM-Ware V-Sphere 6.5.0, based on VMware ESX, an enterprise-class type-1 hypervisor that includes and integrates vital OS components, such as a kernel;
- *90 virtual processor*;
- 267 GB of RAM;
- 1.473 GB of storage.

Upon this computing infrastructure, several virtual machines (VMs) have been created for CLASS, such as:

- One VM for the Open Street Map (OSM) platform (named *openstreet*), with a Linux Ubuntu 16.04;
- One VM for the LoRa server platform (named *Loraserver*), with a Linux Debian 9;
- Four VM for the OpenWhisk cluster (named *CLASSibm1, CLASSibm4, CLASSibm5, CLASSibm6*);
- Four VM for the first Kubernetes cluster (named *kube4*, *kube5, kube6* and *kube7*), with Linux Centos7.

- Four VM for the second Kubernetes cluster (named *kube8, kube9, kube 10 and kube 11)*, with Linux Ubuntu 18.04.

The detailed list of VMs dedicated to the implementation of the CLASS use cases is given in Table 2

Table 2. List of CLASS VMs and their resources in the Modena data center

Virtual machine name	Operative system	Number of CPU	RAM	Hard disks
CLASSibm	CentOS 7	8	64 GB	3 (196 GB)
CLASSibm1	Ubuntu 18.04	16	16 GB	3 (196 GB)
CLASSibm4	Ubuntu 18.04	8	16 GB	1 (100 GB)
CLASSibm5	Ubuntu 18.04	8	16 GB	1 (100 GB)
CLASSibm6	Ubuntu 18.04	8	16 GB	1 (100 GB)
Loraserver	Debian 9	4	2 GB	1 (16 GB)
Kube4	Centos 7.5	2	16 GB	2 (96 GB)
Kube5	Centos 7.5	1	8 GB	2 (96 GB)
Kube6	Centos 7.5	1	8 GB	2 (96 GB)
Kube7	Centos 7.5	1	8 GB	2 (96 GB)
Kube8	Ubuntu 18.04	4	8 GB	1 (60 GB)
Kube9	Ubuntu 18.04	4	8 GB	1 (60 GB)
Kube10	Ubuntu 18.04	4	8 GB	1 (60 GB)
Kube11	Ubuntu 18.04	4	8 GB	1 (60 GB)
miniKube	CentOS 7.5	16	64 GB	2 (96 GB)
openStreetMap	Ubuntu 16.04	1	1 GB	3 (45 GB)

3 Collision detection use case evaluation

3.1 Description

The aim of the collision detection use case is to identify situations that may lead to collisions between road users and generate alerts to warn the involved vehicles in time. In this section, we will provide the end-to-end evaluation of the collision detection use case, from two perspectives:

- ***Analytics perspective***: The performance of the analytics methods employed in CLASS will be evaluated. Even though the aim of CLASS has not been the optimization of the data analytics methods per se, their capability to meet the requirements of the use case has been validated.
- ***Computation perspective***: The end-to-end performance of the data analytics workflows executed across the compute continuum has been evaluated, showing capabilities and limitations of the complete CLASS workflow in real-life scenarios where real-time requirements need to be met.

3.1.1 Collision detection use case evaluation from an analytics perspective

This section presents the evaluation of the collision detection use case focusing on the performance of the data analytics for the timely detection of collisions in real-life scenarios. The methodology selected for this evaluation is described next.

In order to obtain some meaningful results over the analytics performance, two reference videos recorded from two different cameras at the MASA (with IDs 6310 and 20939) have been selected. In both videos, a collision scenario has been set up. The target of this evaluation is twofold: i) to determine the capability of the CLASS data analytics to timely detect the hazardous situations, and ii) to examine how the analytics performance is affected as the overall computation load of the computing infrastructure is increased.

As a first step, an ideal (and not realistic) baseline scenario has been determined for each reference video: for each camera frame, there exist sufficient computing resources to timely execute all data analytics involved in the collision detection use case, i.e., from object detection to trajectory prediction and collision detection. In continuation, the analytics workflow have been executed over the same video on the MASA infrastructure using the CLASS Software Architecture (SA), while additional background data for live camera feeds execute simultaneously.

3.1.1.1 Baseline scenario 1: parking exit at the Via Pico della Mirandola roundabout [camera 6310]

The first baseline scenario is described in Figure 22. The reference video has been recorded by camera 6310, capturing the parking exit towards the Via Pico della Mirandola roundabout. The video (which from now on will be referred to as 6310-video) captures a vehicle exiting the parking, while another car is approaching the roundabout. In this staged scenario, both cars are approaching with high speed and stop abruptly before impact, thus simulating a potential collision scenario.

For the baseline evaluation, the video has been fully processed in three steps: i) the object detection and tracking COMPSs-based workflow has been executed on every video frame, populating the DKB with all the obtained information (see Section 1.2.6), ii) TP has been invoked for all objects with a sufficient history of positions (see Section 1.2.5), and iii) CD has been executed for all objects with calculated trajectories.

Figure 22. Aerial view of the parking exit at the Via Pico della Mirandola roundabout – point of view of camera 6310

A visualization tool designed in CLASS has been employed for the representation of the outcome of the data analytics on each of the input video frames. The information contained in the visualization output (as shown for example in Figure 23) is the following:

- The *bounding boxes* are the outcome of the object detection method. Different colors are used for each type of detected object, e.g., blue for cars, orange for pedestrians, purple for bicycles, etc.
- The *green dots* behind each detected objects and the object IDs are the output of the tracker, showing the history of positions where the object has been detected in previous frames

- The red lines correspond to the predicted trajectories of the objects with the necessary number of historical positions. In this examples, 8 points are predicted with a 500 ms interval between them (depicted as red dots).
- *Collision alerts*, if present, are depicted as black dots. Furthermore, the list of detected collisions and the ids of the involved road users are also printed in the leftmost part of the frame.

Based on the video, it has been determined that the staged collision between the two vehicles take place at frame 249, as shown in Figure 23. The two involved vehicles are the CLASS Maserati car, identified in the video as object 6310_337, and another Maserati car with id 6310_335. The timestamp of this event has been used as a reference for the calculation of the time margin between the detection of collisions based on the analytics and the actual collision.

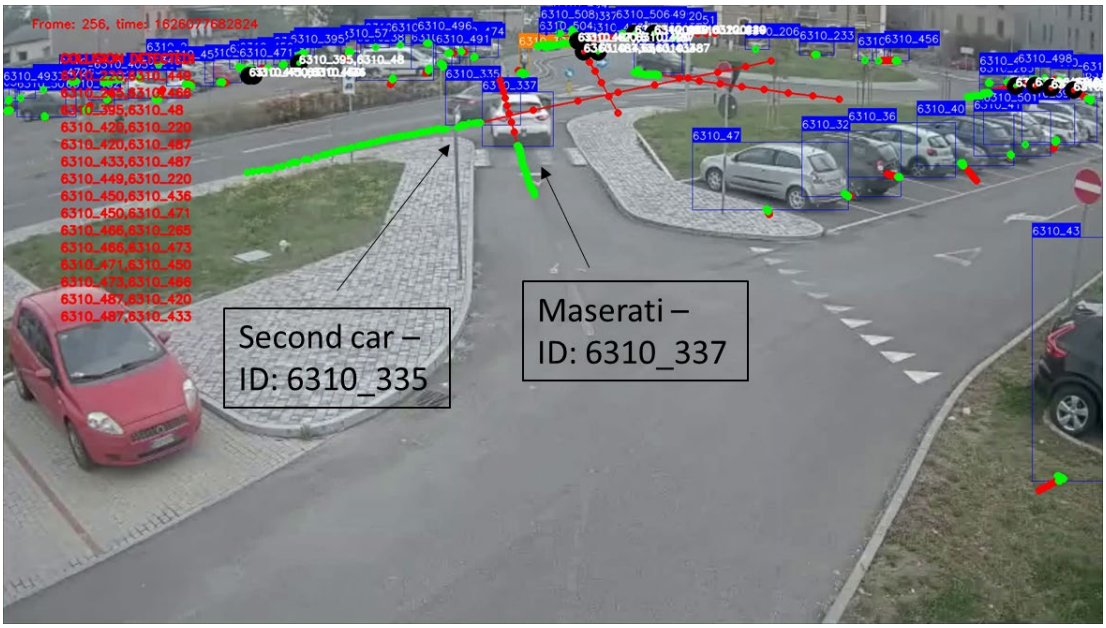

Figure 23. Moment of collision @ frame 256 of the recorded video from the 6310 camera

The first metric that has been measured was the time margin between the generation of the first collision detection alarm and the moment of collision. For the baseline scenario, the first collision detection for the two vehicles has taken place in frame 195, as depicted in Figure 24, corresponding to a 2.25 time margin.

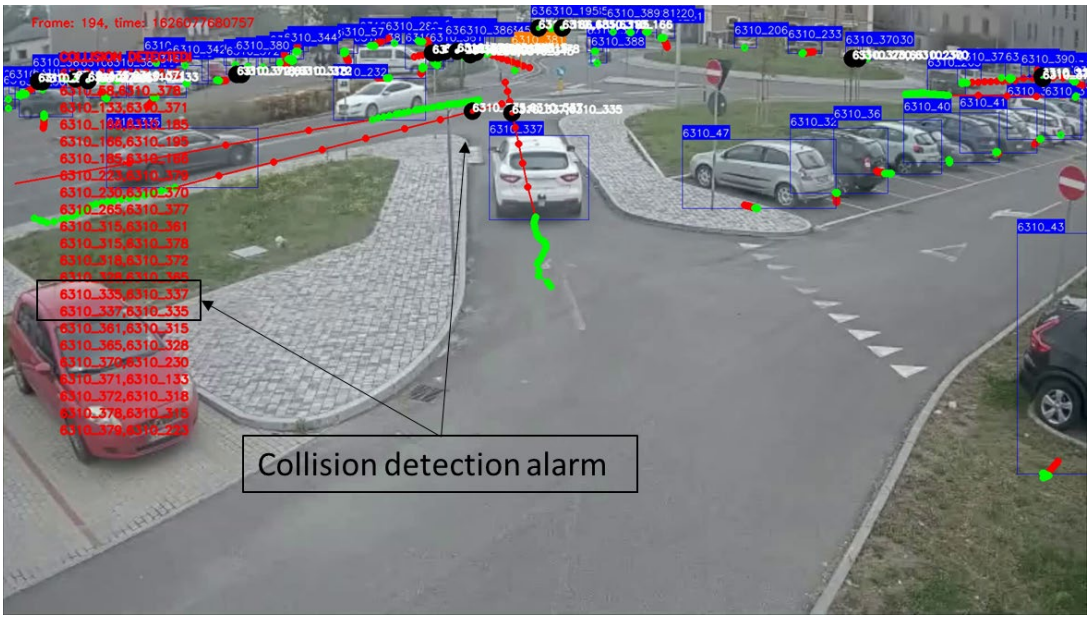

Figure 24. Snapshot of first detection of the collision between the two vehicles @ frame 194

It should be noted that, in each frame, the CD calculates the potential collisions for each object with a trajectory prediction against all other objects present in the frame. As a result, for each pair of vehicles, each collision is computed twice, i.e., for the pairs (object_1, object_2) and (object_2, object_1). These two calculations are not symmetric due to the implementation of the CD (with the definition of the surrounding areas, see Section 1.2.7), but as a general rule both calculations tend to provide a similar output (i.e., detection of potential collision or not).

For the baseline scenario, the focus is laid on the potential collisions detected between the two reference vehicles, with ids 6310_337 and 6310_335. Figure 25 shows the number of generated collision alerts (as blue dots) generated for the pair (6310_337, 6310_335) and the time margin with respect to the moment of actual collision, corresponding to t=0 (marked as an orange diamond). Exactly the same number of warnings have been also generated for the pair (6310_335, 6310_337), but are not depicted in the figure.

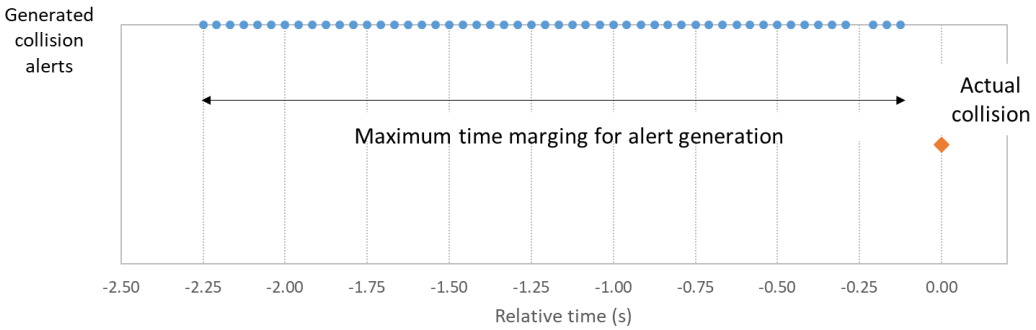

Figure 25. Generated collision alerts between the reference vehicles (6310_337, 6310_335) in baseline scenario for 6310-video

During the processing of the video, the CD analytics return a higher number of generated warnings, which usually correspond to false alarms that may be caused by errors of the detection and tracking analytics (e.g., parked cars may seem to slightly move in consecutive frames, triggering a collision alert), or they may refer to events that take place in the background of the frame but are very hard to evaluate (due to lack of clear visibility). To simplify the analysis, all collision alerts not involving the two reference vehicles have been considered as false alarms, producing computational overhead that does not contain any useful information. A summary of the obtained data describing the baseline scenario for the 6310-video are given in Table 3.

Table 3. Summary of obtained data for 6310-video baseline scenario

Collisions detected between pair (6310_338, 6310_335)	56
Collisions detected between pair (6310_335, 6310_338)	54
Maximum time margin (s) (first collision detection wrt. actual collision	2.25
Total number of useful collisions	110 (11.7%)
Total number of overhead alarms	830 (88.3%)
Total generated alerts	940

3.1.1.2 Baseline scenario 2: Str. Attiraglio and Canaletto Sud roundoubout [camera 20939]

A similar analysis has been conducted for another reference video recorded by camera 20939, covering a part of Str. Attiraglio towards the Canaletto Sud roundabout, as shown in Figure 26. The main staged hazardous event in this video is the crossing of a bicycle (object 20939_240), while a CLASS Maserati car is approaching (object 20939_84). For this scenario, the moment of collision has been set to frame 263, as depicted in Figure 27. Again, the timestamp of this event has been used as a reference for the calculation of the time margin between the detection of collisions based on the analytics and the actual collision.

Due to a last-minute issue on the execution of the data-analytics workflow on the scenario 2, the real-time processing on this scenario is not included in this deliverable. We however consider that there is still value to include this information in this document. This scenario remains as a future work.

Figure 26. Aerial view of the Str. Attiraglio and Canaletto Sud roundabout – point of view of camera 20939

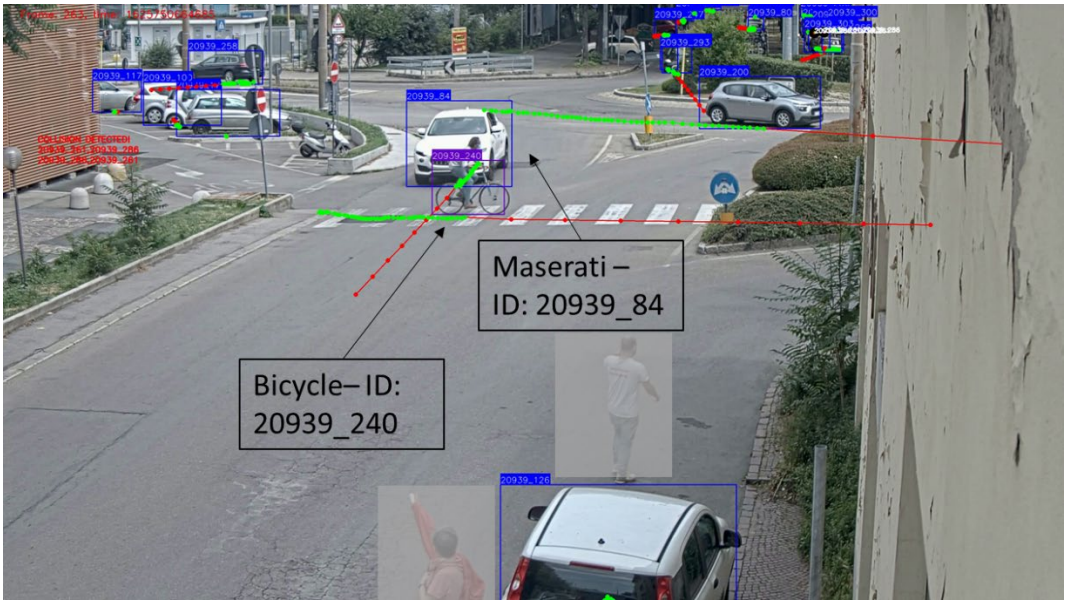

Figure 27. Moment of collision @ frame 263 of the recorded video from the 20939 camera

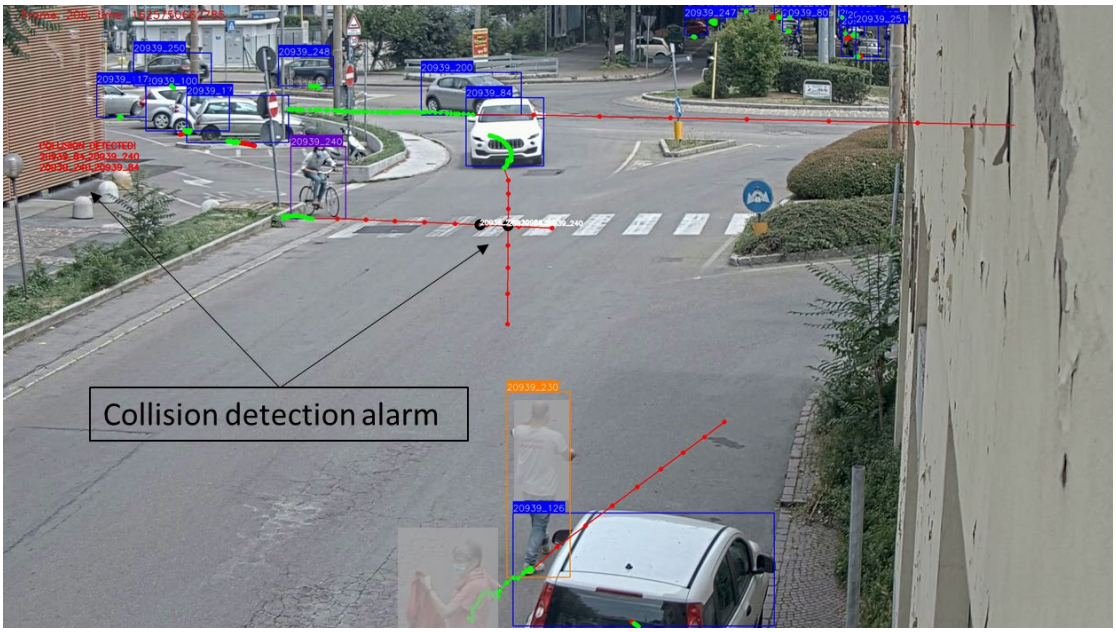

Figure 28. Snapshot of first detection of the collision between the two vehicles @ frame 195

In this case, the first detection of collision, triggering an alert, has taken place at frame 206, as depicted in Figure 28, corresponding to a ***time margin of 1.9 seconds***.

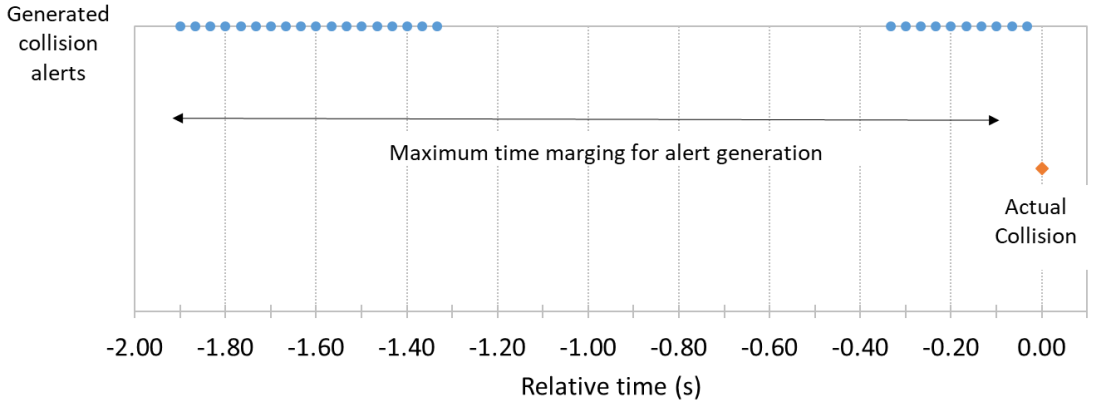

Figure 29. Generated collision alerts between the reference vehicles (20939_84, 20939_240) in baseline scenario for 6310-video

An interesting point is that an onset of collision alerts is triggered as soon as the bicyde obtained a predicted trajectory (starting from frame 206), but then no more collisions are detected until some milliseconds before the collision, as shown in Figure 29. This can be explained by considering that in the staged scenario, the car actually started braking when approaching the bicycle. This change in speed has been reflected in the predicted trajectory, where the predicted path of the car is significantly shorter. This can be verified by comparing the trajectory depicted in Figure 30 (frame 239, 0.8 s before the collision) with the one in Figure 28. Once the bicycle passes in front of the car, the car starts accelerating again (see for example the trajectory in Figure 27), triggering another onset of alerts.

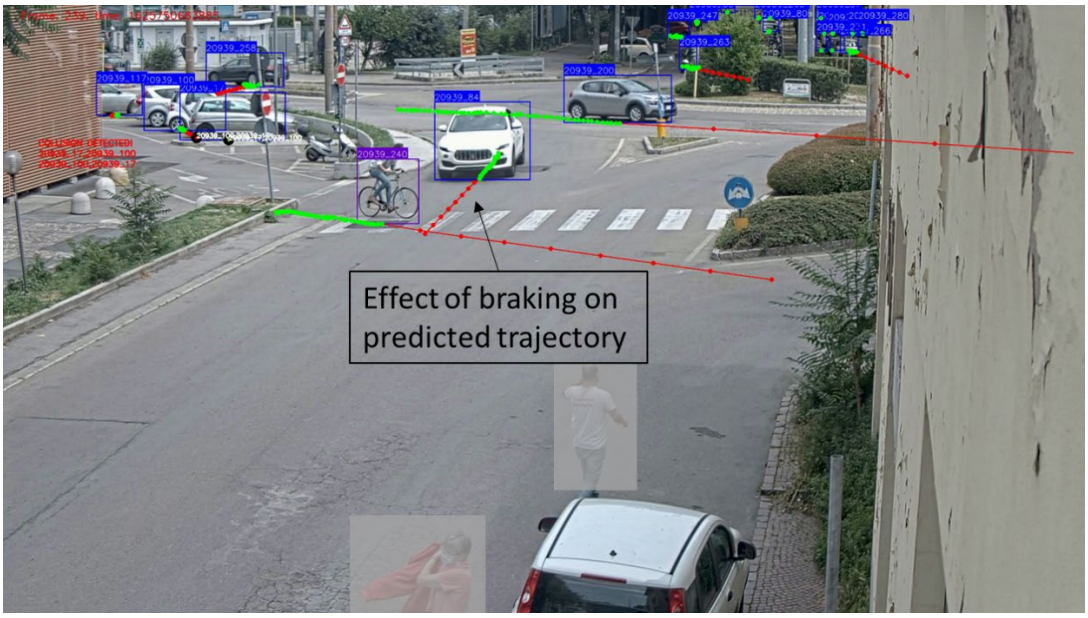

Figure 30. Example of how the braking of the car affects the predicted trajectory

A summary of the obtained data describing the baseline scenario for the 20939-video are given in Table 4.

Table 4. Summary of obtained data for 20939-video baseline scenario

Collisions detected between pair (20939_84, 20939_240)		28
Collisions detected between pair (20939_240, 6310_84)		25
Maximum time margin (s) (first collision detection wrt. actual collision		1.9
Total number of useful collisions		53 (5.64%)
Total number of overhead alarms		887 (94.36%)
Total generated alerts		940

3.1.1.3 Real-time processing of the 6310-video (4 video sources)

In continuation, the same 6310-video has been processed in real-time, with additional traffic generated from three more video sources (i.e., live streams from cameras covering the MASA). The frames from all cameras are received and processed concurrently by the CLASS analytics workflow. In this scenario, the two cars under study correspond to the detected objects 6310_314 (CLASS Maserati) and 6310_312.

In this execution, the collision takes place in frame 257[10] (corresponding to the same relative position as in the baseline scenario).

Due to the additional generated data and the constraints imposed by the limited resources available at the edge, not all frames can be processed by the CLASS SA. It should be noted that the total frames per second processed by the CLASS SA are maintained the same (around 10 fps), but the processed frames per video source is roughly divided by four (given the four connected sources).

Furthermore, the higher number of detected and tracked objects from all cameras significantly increase the computation needed for the TP and CD calculations, as will also be commented in Section 3.1.2.

Due to all the above considerations, the performance of the analytics is affected in the following way. Since less frames from the 6310 are processed and the TP and CD calculations take longer to complete, the number of detected collisions between the two cars is lower with respect to the baseline scenario. In this case, the detection of the frame collision takes place at frame 209, as shown in Figure 31, and only three collision alerts are generated, as depicted in Figure 32. However, the first alert still is generated with an acceptable margin of 2 seconds, showing the validity of the data analytics executed in real time over the CLASS SA.

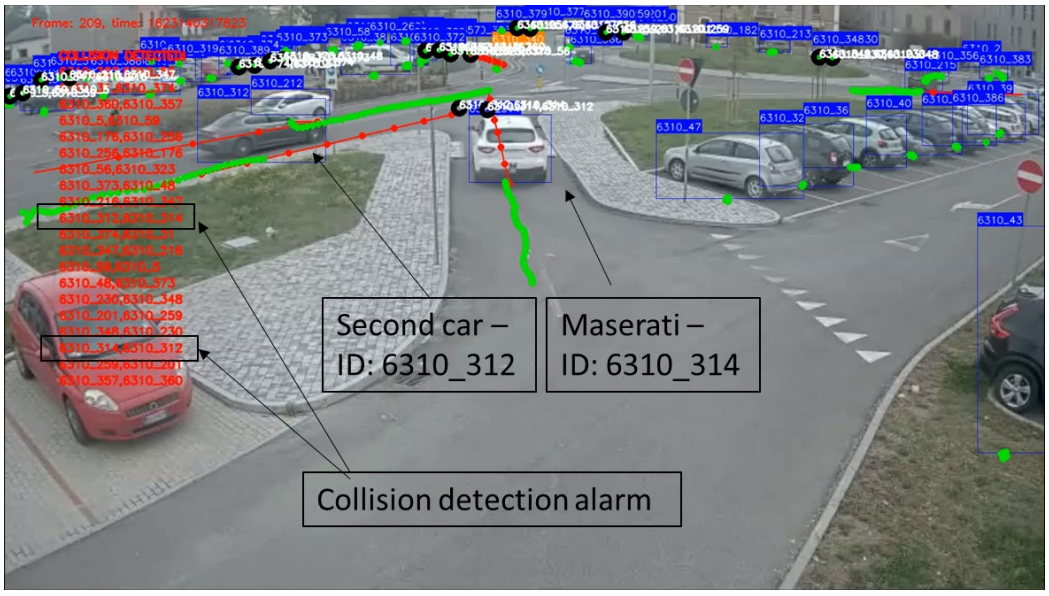

Figure 31. Snapshot of first detection of the collision between the two vehicles @ frame 209

[10] The analysis of the 6310-video in real-time with additional connected sources affects the numbering of the frames. However, this is not important since all the analysis is based on the obtained timestamp between processed frames.

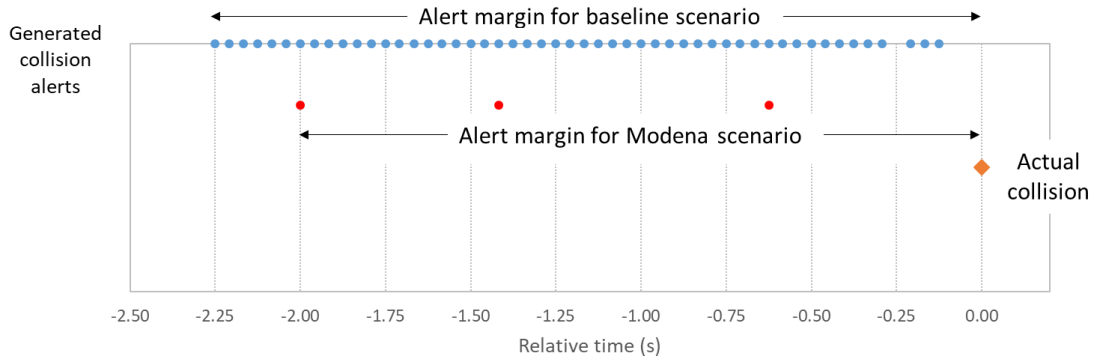

Figure 32. Generated collision alerts between the reference vehicles for the real-time scenario with 4 video sources (red dots) and the baseline scenario (blue dots) for the 6310-video

A summary of the obtained data is given in Table 5. The very high number of overhead alarms is also due to the data obtained from the three additional video sources.

Table 5. Summary of obtained data for the real-time execution of the 6010-video scenario, considering 4 video sources

Collisions detected between pair (20939_84, 20939_240)	3
Collisions detected between pair (20939_240, 6310_84)	3
Maximum time margin (s) (first collision detection wrt. actual collision	2
Total number of useful collisions	6 (1.49%)
Total number of overhead alarms	396 (98.51%)
Total generated alerts	402

3.1.1.1 Real-time processing of the 6310-video (8 and 14 video sources)

Finally, the same experiment has been repeated by increasing the number of total video sources to eight and fourteen (i.e, the reference 6310-video plus 7 and 13 live streams respectively, from cameras in the MASA). In both scenarios, it has not been possible to obtain valid results for the analytics, due to the excessive load of processed data which could not be handled by the available computing resources.

The next two figures show the behavior of the data analytics in the 8 camera sources scenario. Figure 33 shows the available data as the two cars are approaching the intersection. We can observe that the two reference vehicles have been detected and tracked, but there are not sufficient positions for the execution of the TP. On the contrary, in the previous experiment with only 4 video sources, the TP was obtained, as shown by the red trajectory lines in Figure 31.

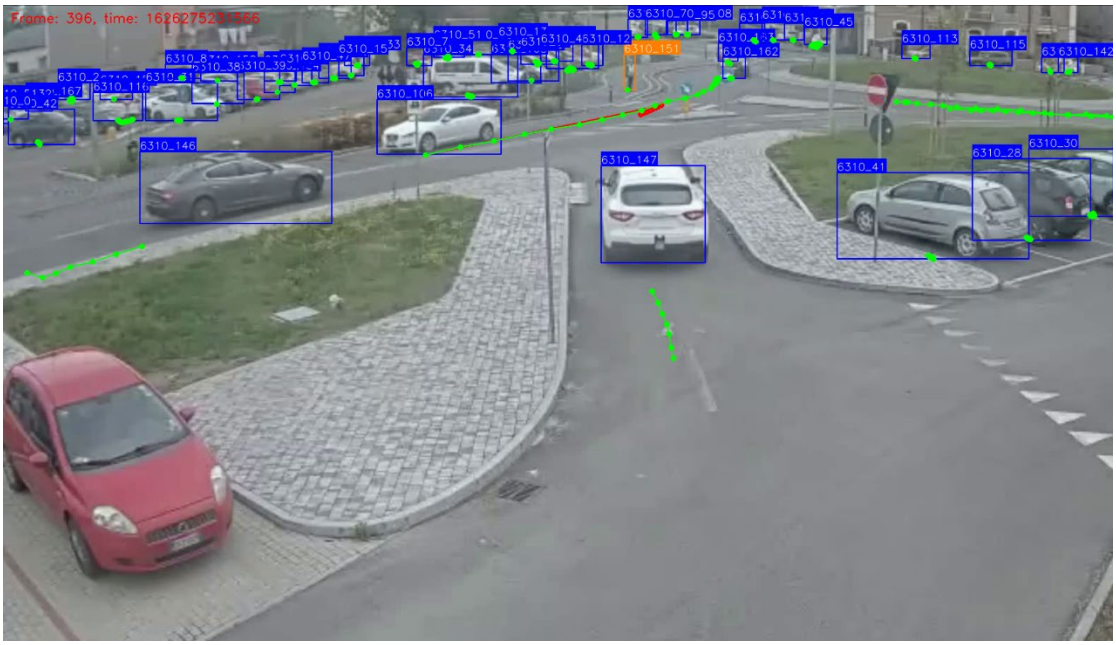

Figure 33. Snapshot from the execution of the 6310-video with 7 additional video sources. The TP has not been yet calculated for the reference vehicles.

Figure 34 shows the evolution of the experiment, when the two cars have almost arrived at the intersection. At this point, the TP has been invoked for the two cars and a TP has been calculated (red lines). However, it can be observed that the starting point of the predicted trajectory does not correspond to the current position of the cars, but to a previous position.

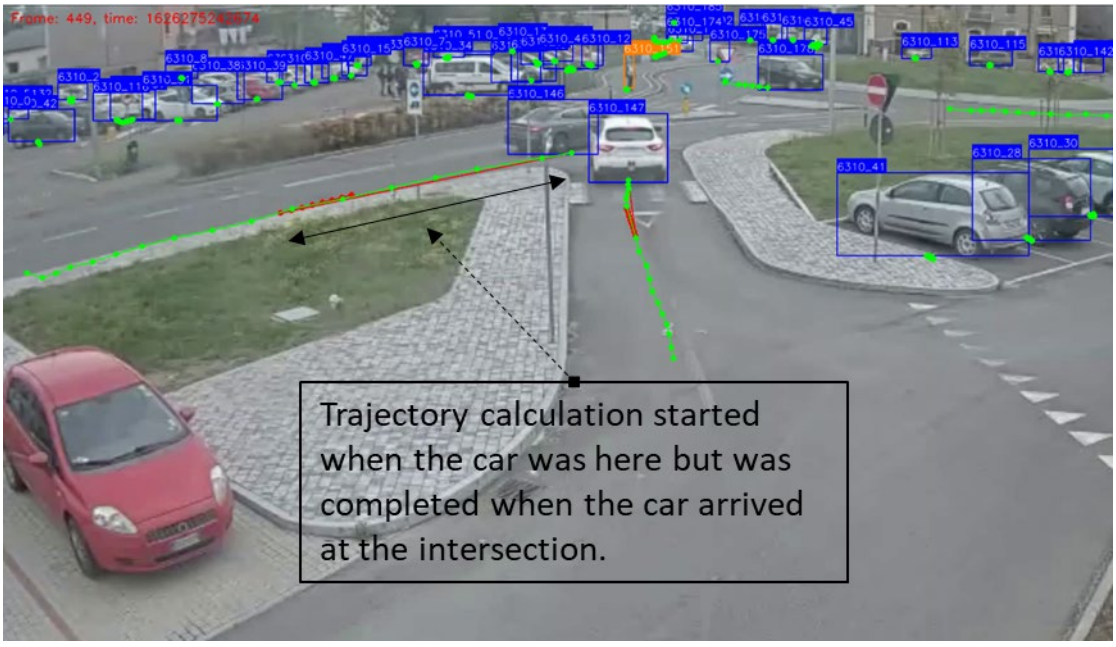

Figure 34. Impact of the increased system load (8 video sources) on the completion of the trajectory prediction calculation

The experiment when considering 14 cameras presents a behaviour similar to the one observed in the 8 camera scenario.

Next section evaluates the end-to-end response time, considering the computing resources across the compute continuum, i.e., from edge to cloud, that explains the results presented in this section.

3.1.2 Collision detection use case evaluation from a computation perspective

In this section, the performance evaluation of the collision detection use case is presented from a computation perspective, taking into account the computing resources available across the compute continuum. To do so, the complete data analytics workflow, as presented in Figure 1, has been executed over the CLASS SA, and the end-to-end execution time has been obtained for four different experiments: varying number of video sources, with 1, 4, 8 and 14 cameras. For all experiments, the 6310-video has been used as the first video source, and live feeds from other cameras at the MASA have been used to increase the computational load of the system.

Figure 35 shows the obtained end-to-end execution times for the four experiments, calculated as the sum of the execution times of the different components of the workflow executed across the compute continuum. The information is also included in Table 6. Specifically, the end-to-end execution time is decomposed into five components:

i) The ***execution time of the COMPS-based analytics workflow executed at the edge*** (i.e., at the four fog nodes at the MASA). The edge workflow consists of the methods for the object detection, tracking, deduplication and data federation.

ii) The ***execution time of the trajectory prediction and collision detection at the cloud***. This time is affected by the number of detected and tracked objects.

iii) The ***time consumed for the retrieval and update of objects stored the DKB*** (implemented with dataClay) at the cloud, requested by the trajectory prediction.

iv) The ***time consumed for the retrieval of objects with valid trajectory predictions by the DBK*** at the cloud, requested by the collision detection method.

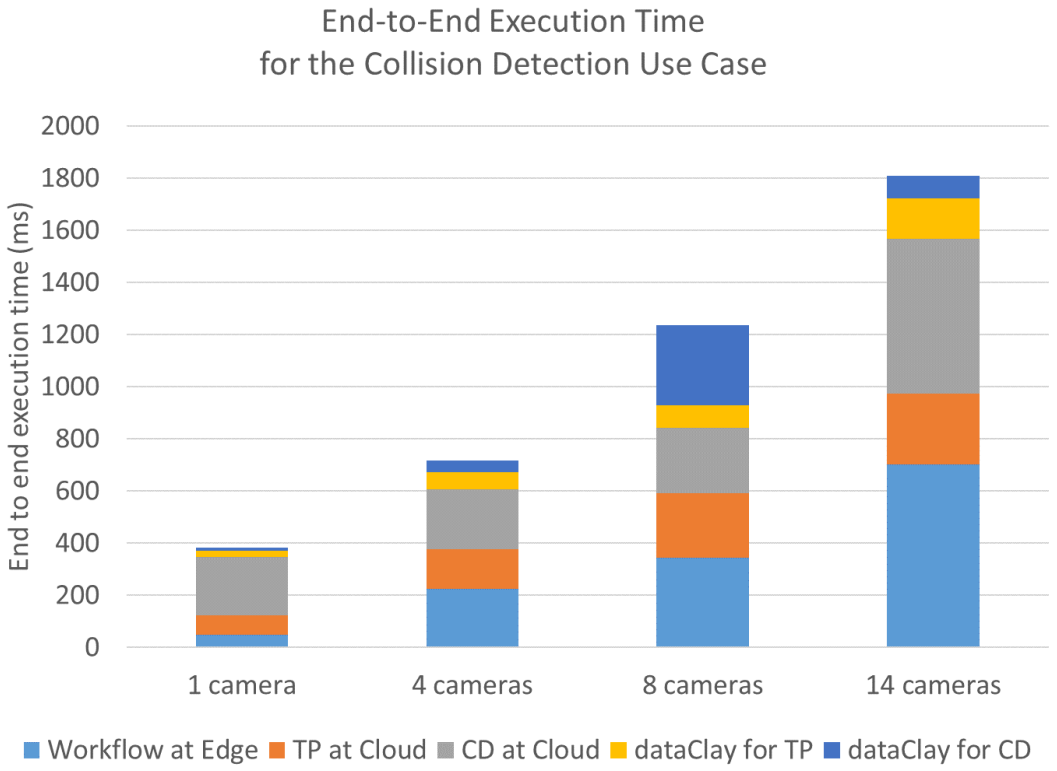

Figure 35. End-to-end execution times of the complete CLASS data analytics workflow for the collision detection use case

Table 6. Decomposition of the end-to-end execution times for the collision detection use case

	Execution times (ms)					
Video-sources	**Workflow at Edge**	**TP at Cloud**	**CD at Cloud**	**dataClay for TP**	**dataClay for CD**	**End-to-end execution**
1 camera	47.0	76.4	224.4	21.8	12.2	381.8
4 cameras	225.4	152.4	228.8	64.5	44.5	715.6
8 cameras	343.8	247.4	251.1	85.4	308.5	1236.2
14 cameras	701.9	272.1	591.7	155.2	89.4	1810.2

As shown in Figure 35 and Table 6, the execution time of the edge analytics is affected by the increasing load, varying the total number of processed frames, i.e., from 10 fps in case of 1 camera, to 2 fps in case of 14 cameras. Overall, the execution of the

complete workflow when considering 8 and 14 cameras, suffers from excessive delays. The impact of these delays on the performance of the analytics was shown in the previous section, in Figure 34, where due to the increased time required for completion for the TP, the obtained data from the computation were outdated.

One interesting observation is that the TP and CD are not invoked by the COMPSs-based edge workflow, but are executed periodically and in an asynchronous way. As a result, the end-to-end execution time presented in Figure 35 has been calculated as the sum of the average time required by the different components. However, it should be noted that TP and CD are invoked as serverless actions over a varying number of objects, and their execution time is heavily affected by the number of objects. Hence, when zero objects are returned by the TP or CD dataClay calls (e.g., when there are no objects with sufficient positions for the TP to be invoked or there are no moving vehicles detected), the execution time of TP and CD is very low (approximately 56 ms in average). However, at the most critical parts of the reference videos, when the number of detected objects is high, the TP and CD calculations, as well as the corresponding dataClay calls, require a considerably higher amount of time, which is responsible for the deterioration of the quality of analytics.

Concretely, for 1 and 4 cameras, this time remains relatively low, below 200 ms. However, for 8 and 14 cameras, the number of objects increases significantly, causing a bottleneck in the execution of the TP by tripling the TP time. In case of the collision detection, this time is also increased for a higher number of objects with the trajectory predicted.

To highlight this fact, Figure 36 depicts the end-to-end execution time of the analytics, taking into account only the TP and CD actions with non-zero input objects. In this plot, the bottleneck created for the increasing number of objects, especially for the TP method (both the TP calculation and the TP-related dataClay calls) is evident. For reference, the values are included in Table 7.

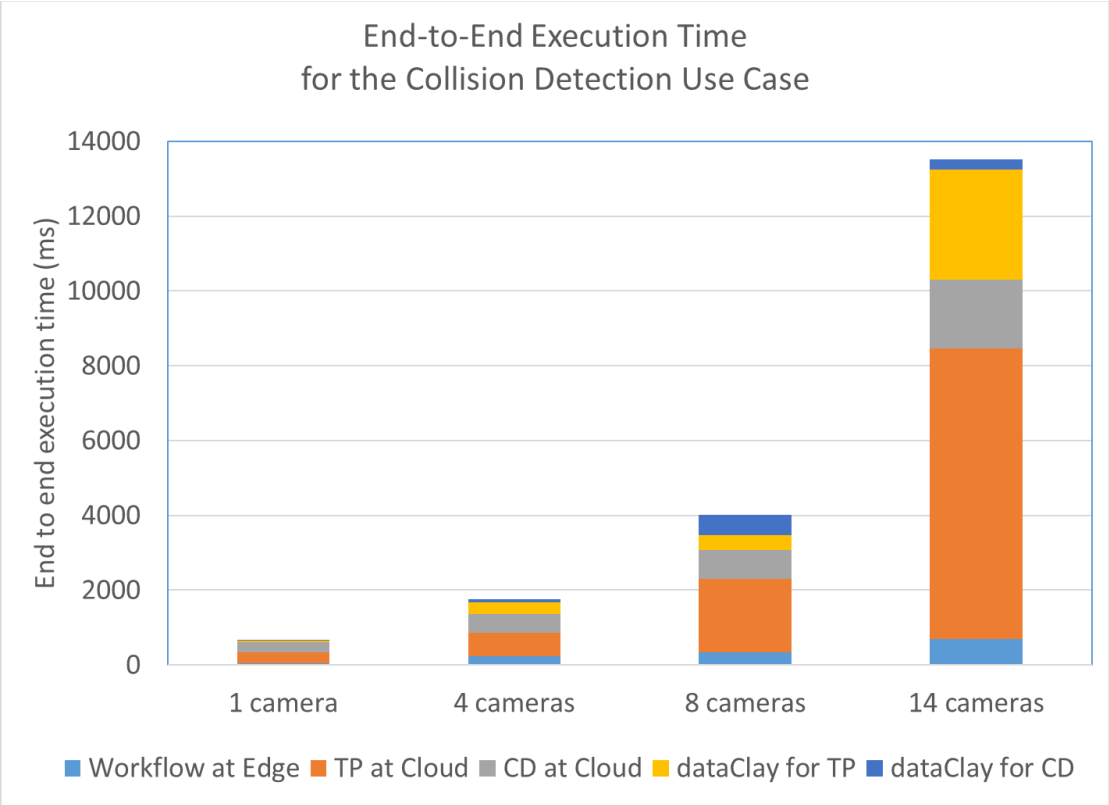

Figure 36. End-to-end execution times of the complete CLASS data analytics workflow for the collision detection use case, when TP and CD execution times are calculated only over non-empty serverless actions.

Table 7. Decomposition of the end-to-end execution times for the collision detection use case, when TP and CD execution times are calculated only over non-empty serverless actions

	Execution times (ms)					
Video-sources	**Workflow at Edge**	**TP at Cloud**	**CD at Cloud**	**dataClay for TP**	**dataClay for CD**	**End-to-end execution**
1 camera	47.0	284.0	282.0	29.9	15.0	657.7
4 cameras	225.4	624.8	521.4	298.7	88.3	1758.7
8 cameras	343.8	1952.7	770.5	409.0	530.5	4006.4
14 cameras	701.9	7765.5	1842.5	2927.9	274.0	13511.7

Finally, Table 8 presents two additional metrics for the evaluation of the TP and CD performance. The two columns at the left show the average and maximum (in

parenthesis) number of objects considered in the TP and CD serverless actions. It can be seen that the number of objects generally increases as more video sources are added. At the rightmost side, the average/maximum execution time of TP and CD actions weighted by the number of considered objects is included.

It can be seen that this metric is not directly proportional to the number of object: for instance the *TP/Object* time for 4 cameras is higher wrt. to the case of 8 cameras). This phenomenon can be caused by some initial Lithops actions being executed in cold start, causing increased delay even for a low number of objects.

Table 8. Average and maximum values of i) objects per TP and CD serverless action and ii) execution time for TP and CD per object

Video-sources	Average (Max) Number of Objects per action		Average (Max) execution time (ms) per Object	
	TP	CD	TP/Object	CD/Object
1 camera	19.9 (27)	8.1 (12)	14.4 (12)	31.6 (61.3)
4 cameras	15.2 (108)	30.5 (42)	53 (307.4)	18.6 (136.3)
8 cameras	149.6 (179)	30.2 (51)	12.9 (77.3)	22.6 (240.7)
14 cameras	245.4 (304)	60 (121)	30.8 (92.8)	16.9 (299)

As a conclusion, three key improvements could be applied as a future step to overcome the bottleneck in the performance of TP and CD are:

1. Guarantee that all Lithops functions are warmed, so no cold start is suffered.
2. Enhance the filtering mechanism of objects for which the TP and CD are applied (e.g., by limiting the computation to only specific parts of the frame).
3. Reduce the time required to access the DKB, by improving the retrieval mechanisms or using a different approach to the data storage and transfer.

3.2 Air pollution estimation use case

3.2.1 Background on vehicle-related air pollution models

This section presents the evaluation of the air pollution estimation use case focusing on the performance of the data analytics for the calculation of the vehicle-related air pollutants based on real-time data.

In order to interpret the obtained results, some background on the methods used to estimate the level of vehicle-related pollution is needed. Accurate measurements of the pollution levels due to the circulation of vehicles are very hard to obtain, due to the presence of multiple contamination sources within the city (e.g., emissions from factories, buildings, large communication infrastructures such as airports and harbors). As a result, the main approach is the use of air pollution estimation models,

such as COPERT[11] (an EU standard vehicle emissions calculator) or MOVES[12] (a similar approach adopted in the USA). Such solutions typically use long-term statistics on the vehicle population, mileage and speed, as an input to obtain pollution levels at a macroscopic level.

More recently, solutions such as PHEMLight[13] have been developed [7], enabling the estimation of pollutants at a microscopic level (e.g., at a per-hour basis), together with more detailed traffic simulation models that capture the traffic behavior at given urban areas [8].

In CLASS, the novelty lies in using actual traffic information obtained in real-time, through the detection and tracking methods of the CLASS data analytics workflow. This approach has two key advantages: i) it relies on real traffic information (i.e., vehicle type, position and speed/acceleration), instead of simulated traffic models, and ii) it enables the estimation of air pollutant levels at a much smaller scale (i.e., in the order of minutes, or even seconds). As a result, more insights can be gained with respect to the impact of real-life traffic behavior on the level of emissions.

3.2.2 Analytics perspective

For this use case, a recorded video from the 20936 and 20937 cameras at the MASA area has been used. This video has been selected because it is one of areas of MASA with a higher number of vehicles (see Figure 37).

Figure 37. Snapshot from the recorded video from the cameras 20936 (left) and 20937 (right), used for the air pollution use case evaluation

For the evaluation of the air pollution use case, the CLASS analytics workflow has been executed for a total of 75000 iterations, corresponding to the processing of approximately 42 minutes of the recorded video. Four different experiments have been conducted for this time interval, changing the frequency of invocation of the PHEMLight model:

i) Execution of the PHEMLight model on the data generated approximately every 2 minutes, resulting to 15 invocations.
ii) Execution of the PHEMLight approximately every 8 minutes, resulting to 5 invocations over the 42 minutes interval.

[11] https://www.emisia.com/utilities/copert/
[12] https://www.epa.gov/moves
[13] https://sumo.dlr.de/docs/Models/Emissions/PHEMlight.html

iii) Execution of the PHEMLight approximately every 13 minutes, resulting to 5 invocations over the 42 minutes interval.
iv) Execution of the PHEMLight at the end of the 42 minutes of video.

At the end of the video, the intermediate results of pollutant emissions computed with the PHEMLight model are combined to provide the overall accumulated pollution over the 42 minutes interval.

Figure 38 shows the estimated accumulated levels of NOx, PM, CO, HC and NO[14] emissions, measured in kg/h, for the four calculation intervals of 2, 8, 13 and 42 minutes. For each experiment, three different vehicle types have been considered, namely car, bus and heavy-duty vehicle (hdv, corresponding to trucks), since the emission levels for each type of vehicle is significantly different.

Two key observations can be made on the obtained results:

1. In the first place, as mentioned before, it is very hard to obtain reference values for the evaluated scenarios, since air quality indicators measured directly by sensors cannot isolate vehicle-related emissions and are typically affected by multiple sources of contamination, especially in industrial urban environments such as the city of Modena. The results presented in this deliverable have been validated by comparing them with the emission values computed from certain areas in the City of Barcelona, using PHEMLight[15]; results are in the same order of magnitude
2. In the second place, as shown in Figure 38, the aggregated level of each pollutant for each type of vehicle is approximately the same, regardless of the calculation interval. This validates the consistency of the calculations, showing the ability of the CLASS analytics workflow to obtain fine grained results within a very small time-scale, with the same accuracy as hour-long observations. This feature opens the road to a level of analysis that was not possible before, since it enables the possibility of studying the impact of real traffic behaviours that cannot be captured by long-term statistical models. As an example, the proposed approach could estimate the emissions from a truck that remains parked with the engine on for several minutes in a given location, or the impact of traffic congestions at street intersections.

The potential of this new approach has been identified in the last stages of the CLASS project and will be further exploited beyond the lifetime of the project. Indeed, BSC is currently employing the air pollution analytics workflow developed in CLASS to obtain

[14] NOx: Nitrogen Oxides; PM: Particulate Matter; CO: Carbon monoxide; HC: Hydrocarbons
[15] These results have een computed by BSC Earth Science Department (https://www.bsc.es/research-development/research-areas/atmospheric-composition/urban-air-quality-modelling)

a fine-grain estimation of vehicle-related pollutants in real-time, in collaboration with the city of Barcelona.

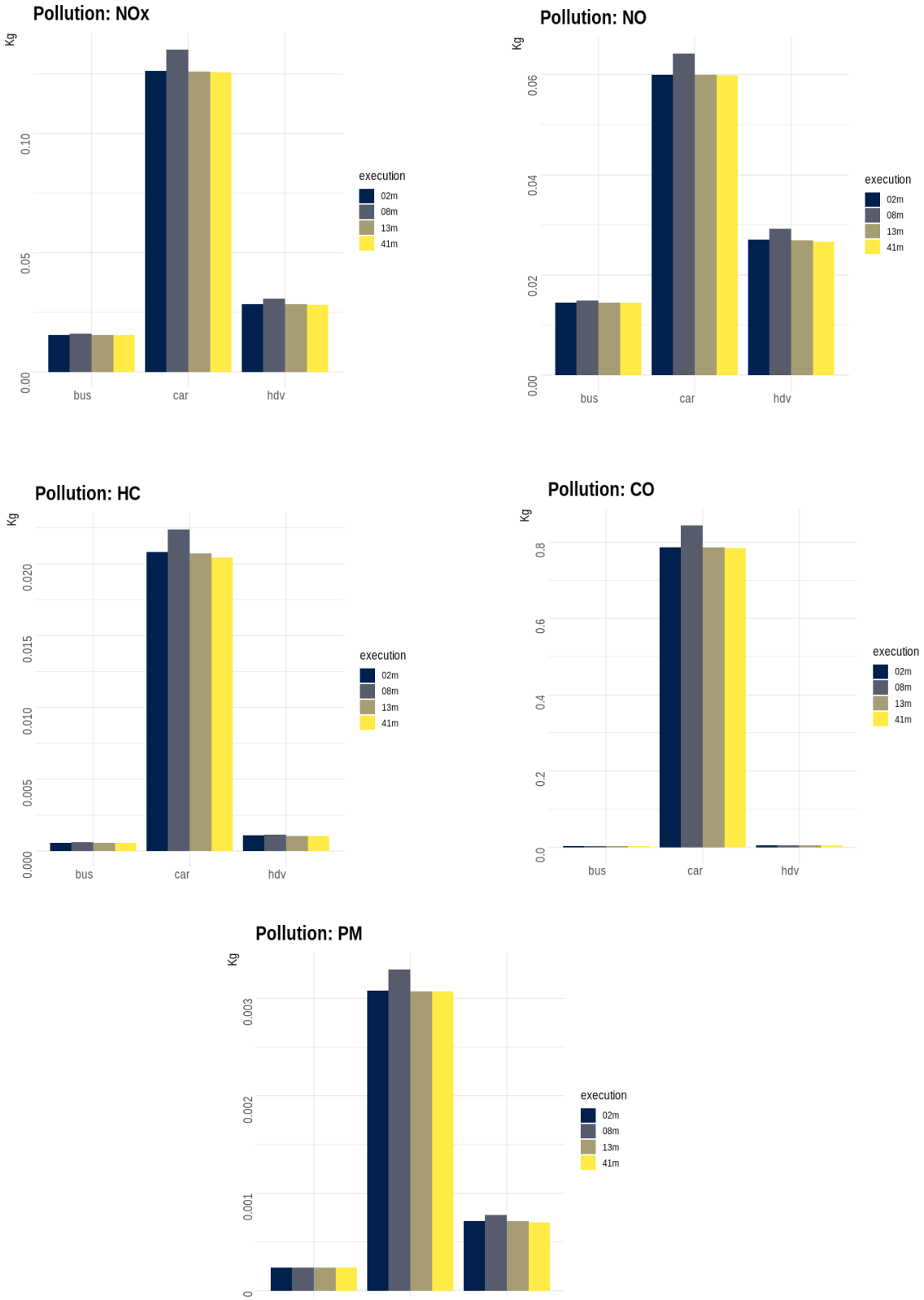

Figure 38. Estimated levels of vehicle-related pollutants (NOx, NO, HC, CO and PM), measured in kg/h, at four different time intervals.

3.3 *Digital traffic signs (green routes)*

To estimate the emergency vehicles response time, 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. 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. It is reasonable to assume that both smart and non-smart vehicles will coexist until next-generation cars will completely overtake the classic transportation means.

To determine the ambulance travel time with an increasing number of smart agents, multiple scenarios were run. In all these scenarios, we report the average travel time over a set of 30 ambulance trips. The first scenario consisted of 12000 agents with a scaling number of smart agents and results are depicted in Table 9.

In Table 9 it is shown 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%. Another key variable in determining the average response time is the population size of road users within the interested area. We performed additional tests by increasing the number of agents to 16 000, hence getting closer to the inevitable congestion of the MASA area we modelled in our simulation. In such a scenario, the response time improvement within a population characterized by 75% of smart agents would range from 4.3% to 8.33% with respect to the baseline scenario of no smart agents at all.

Table 9. Ambulance average travel times with different kinds of agents

	Smart agent percentage	Ambulance travel time [s]	% 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

Acronyms and Abbreviations

ADAS – Advanced driver-assistance systems
API – Application Programming Interface
Caas – Container as a Service
CD – Collision Detection
CLI – Command Line Interface
D – Deliverable
DAG – Direct Acyclic Graph
DKB – Data Knowledge Base
DNN – Deep Neural Network
EXPRESS – EXtended PREdictability ServerlesS
FaaS – Function as a Service
HDV – Heavy Duty Vehicle
IoT – Internet of Things
LiDARs – Light Detection and Ranging
LoRa – Long Range
M2M – Machine to Machine
MASA – Modena Automotive Smart Area
MQTT – Message Queuing Telemetry Transport
MS – Milestone
OSM – OpenStreetMap
RMSE – Root-Mean-Square Error
SA – Software Architecture
SDK – Software Development Kit
SLA – Service Level Agreement
SUV – Sport Utility Vehicle
TP – Trajectory Prediction
QoS – Quality of Service
WP – Work Package

References

[1] W. A. Kay, H. Andreas and K. Nagel, The multi-agent transport simulation MATSim, Ubiquity Press, 2016.

[2] CLASS, "D1.2 - Final release of the Smart City Use-Cases," March 2019.

[3] CLASS, "D1.4 - Final release of the smart city use case," July 2020.

[4] CLASS, "D3.6 - Validation of the CLASS egde computing subsystem," June 2021.

[5] CLASS, "D3.6 - Validation of the CLASS edge computing subsystem," June 2021.

[6] 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.

[7] R. M., D. M., L. R., H. S., H. M., K. D., W. P., W. M., B. R., S. T. and D.-L. J., "Cooperative Self-Organizing System for low Carbon Mobility at low Penetration Rates - COLOMBO: Deliverable 4.3 Pollutant Emission Models and Optimisation," 2014.

[8] D. Rodriguez-Rey, M. Guevara, M. P. Linares, J. Casanovas, J. Salmerón, A. Soret, O. Jorba, C. Tena and C. P. García-Pando, "A coupled macroscopic traffic and pollutant emission modelling system for Barcelona," *Transportation Research Part D: Transport and Environment,* vol. 92, 2021.

[9] CLASS, "D5.4 - Final release of an augmented platform for analytics workloads," June 2021.

[10] CLASS, "D5.5 - Evaluation of the data analytics platform," June 2021.