D3.6 Validation of the CLASS edge computing subsystem

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M42, June 2021
Dissemination Level	PU
Nature	R
Author(s)	Roberto Cavicchioli (UNIMORE)
Contributor(s)	
Reviewer(s)	IBM
Keywords	Analytics, DNN, real-time

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	Roberto Cavicchioli (UNIMORE)	Initial Draft
0.2	Erez Hadad (IBM)	Internal reviewer
1.0	UNIMORE, BSC	Final version, ready to EC review

Table of contents

1 Executive Summary

This document describes deliverable D3.6 “Validation of the CLASS edge computing subsystem”, which focuses on validating the CLASS computing subsystem at the edge, as required by CLASS DoA [1]. The evaluation is based on what already presented in deliverables D3.1 [2], D3.3 [3], D3.5 [4], and other related CLASS documentation. The quantitative evaluation of CLASS analytics presented in this report is based on CLASS use-case workloads, with the full end-to-end evaluation of the CLASS use-case discussed in deliverable D1.6 [5]. Also, this document reports further optimizations and improvements that have been realized as part of the optimizations and additional impact. Together with the rest of MS4 documents, this document concludes the CLASS project as part of milestone MS4, executed in M31-M42.

Unfortunately, since the pandemic affected the work of integration from MS3, this milestone has been subject to constraints that have slowed greatly the progress. We deliver all planned content under some mitigations, as following:

- The real-time tool on the edge prototype is only applied statically and not dynamically. The original estimation of its integration effort was inadequate due to the pandemic situation. The eventual integration of the dynamic real-time tool is still planned for additional value, but may happen outside the project time-frame.

The document lays out as follows. Section 2 details technical improvements and optimizations introduced in CLASS edge components as part of the ongoing integration. Section 3 consists of evaluation of CLASS computing subsystem at the end of the project, based on experimental evaluation.

2 Code Updates

The edge components of CLASS software stack have been improved from MS3, since the integration of them, thanks to the easy deployment given by the dockerization. We will now descript the different edge analytics components: DNN, tracker, sensor fusion and deduplication.

2.1 DNN and tracking

From cameras located in the streets, objects can be detected, classified, and tracked. All the cameras belonging to the infrastructure are supposed to perform those tasks in real-time and send the extrapolated information to a data aggregator. This aggregator will then de-duplicate repeated information and send messages to the connected cars, which will receive only objects that are relevant to their surroundings.

Geo-localization: To perform the above operations, it is necessary to know the real world position of each object detected from the cameras. To do so, an extrinsic calibration of each camera in the MASA has been performed, in order to have a mapping for each pixel in the camera frame to its GPS position on a geo-referenced map.

Real-time requirements: Data is *constantly* being produced and processed and it is extremely important to guarantee that the results are meaningful by the time they are computed. This is especially relevant for the *Obstacle Detection and Tracking* use case since alerts must raise within a time interval that is useful for the driver to react. A reasonable metric, considered in the scope of the CLASS project, is to get updated results at a rate between 10 and 100 milliseconds. Assuming that the maximum speed of a vehicle within the city is 60 km/h, vehicles will advance between 0.17 and 1.7 meters. This level of granularity is enough to implement the proposed use-cases.

The flow of the application

We have implemented all the tasks inside a single application called class-edge for independent testing before the integration phase and we have released the code open source. class-edge takes in input *N_streams* streams and for each of them perform four main steps:

- First, frames are retrieved via the Real-Time Streaming Protocol (RTSP). Given that the image is taken from a camera, it is very likely that it is affected by distortion. To correct that, undistortion, based on the intrinsic calibration of the camera, is applied. Figure 1a shows an original frame while Figure 1b shows the output of undistortion.
- Object detection is then performed on the undistorted image. For this project we picked the tkDNN (already introduced in D1.4 [6]) implementation of Yolov4 [7]. An example is given by Figure 1c.
- The detection gives in output a list of bounding boxes (BBs). For each of them, a single point is picked to represent the whole object, namely the center of the bottom side of the BB. This pixel is converted first, into a GPS position, and then in meters. This is the format required from the next step: the tracker. Indeed, to track and predict the position of the detected objects an Extended Kalman Filter (EKF) [8] on the real-world position of the object has been applied. Objects between frames are matched together only if the class corresponds and their distance is under a user-defined threshold. Tracking is not only used to have a more robust detection, but also to have a history of the objects. The idea of history is given by the lines in Figure 5.4d.
- Finally, the information can be sent both to the data aggregator, in an anonymized form, and to the optional graphical viewer.

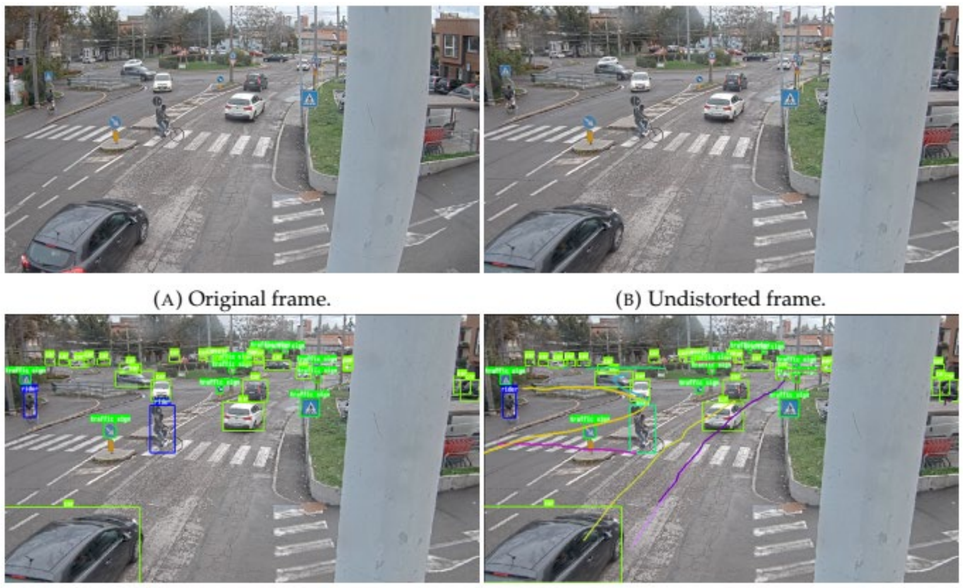

(A) Original frame. (B) Undistorted frame.
(C) Object detection. (D) Object tracking.

Figure 1 : Different steps for the class-edge application

2.2 Sensor fusion

As already presented in D1.4 [6], the sensor fusion task is taken care by coupling different algorithms: the object detection network previously described, a clustering method for the LiDAR points and a similarity fusion one. With respect to the paper referred in D1.4 [9] the performance have improved thanks to the usage of Yolo v4 instead of v3. For an updated comparison of the performances between different versions of Yolo using tkDNN and TensorRT, please refer to: https://github.com/class-euproject/tkDNN.

2.3 Deduplication

As discussed in D1.4 [6], 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.

The simple method described, by searching for all nearest object with the same category of the considered one, was not scalable enough. The deduplication time for just 2 of all the 16 cameras managed by the fog nodes and the smart/connected cars was 12 ms average and 42 ms maximum, which can be considered reasonable, but with 4 cameras it increased to 34 ms average to 106 ms maximum, which is not suitable for our use cases.

We opted for a solution using a knn library (libnabo [10]) that is based on k-d tree to reduce the computational cost. Whit this library the computational time to deduplicate all the 16 cameras and the smart/connected cars was reduced to an average of 0.5 ms and a maximum of 5 ms.

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

3 Evaluation

In this Section we review and evaluate the fit-for-purpose of the CLASS analytics layer. We present a quantitative performance evaluation focusing on both throughput and latency and discuss related observations that have been collected during evaluation.

Following the general approach of this document, we focus on object detection and tracking, sensor fusion and deduplication.

3.1 Evaluation Method

Our evaluation method is designed to be closely aligned with the main CLASS collision avoidance use-case, as following. We use the CLASS deployment in Modena, with class-edge (detection and tracking) and the deduplicator deployed in the fog nodes in the Modena data center, while the sensor fusion is deployed on the smart cars. We use these applications as our benchmark applications. The datasets used for computation are also generated from actual Modena input videos recorded during integration sessions.

Our 4 fog nodes in Modena have 32GB memory each and 6 cores (12 threads) of Intel Core i7 8700K operating at 3.7GHz, equipped also with and NVIDIA TitanV GPGPU. The smart cars have a Drive Pegasus board, which contains 2 NVIDIA Xavier SoC and 2 modified RTX 2060 GPGPUs. Specific tests for the development of the applications might have been done in different hardware (which is descripted if it is the case).

We instrumented the class-edge and deduplicator applications with code that recorded time-stamps of different stages, and used that information to generate detailed profiling information from each invocation. We also measured in a similar way the sensor fusion application.

3.2 Task schematization

In this section we are going to describe our tasks as Direct Acyclic Graphs (DAG) to fit the task model presented in D3.5 [4].

3.2.1 class-edge as DAG

Let us now pick *N_streams* = 2. The heterogeneous conditional DAG of the task is depicted in Figure 2, and its implicit deadline is *D* = 100*ms.*

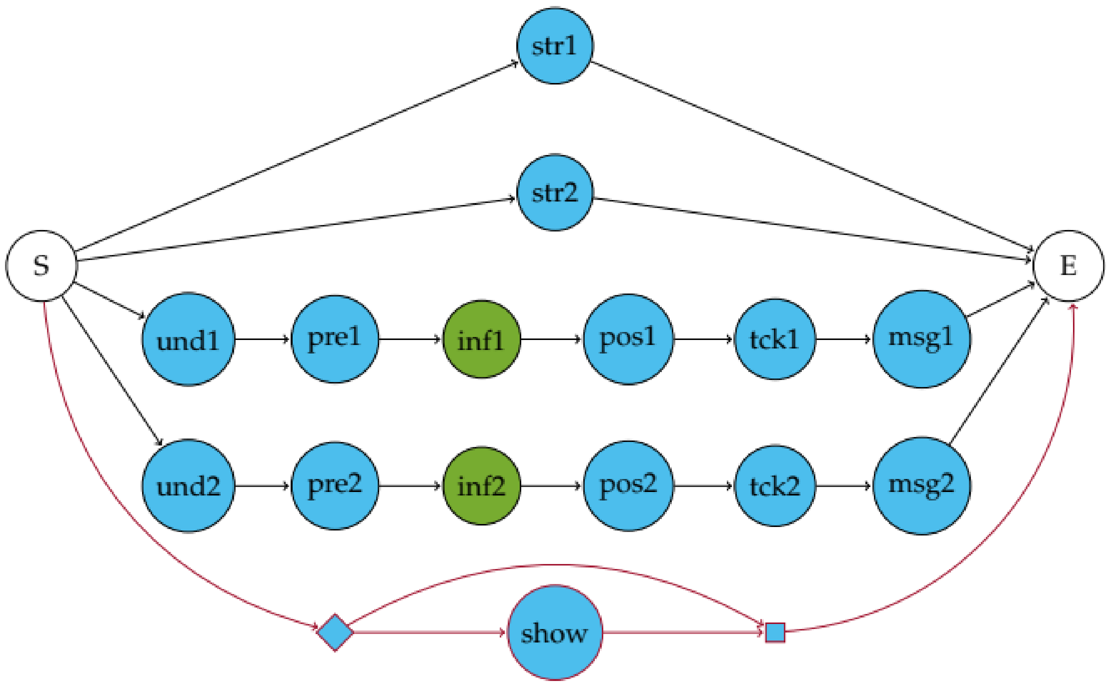


Figure 2 class-edge represented as a Heterogeneous Conditional DAG

The process is actually composed of 5 threads: 2 threads, one for stream, that retrieve the stream ad store the image (*str1/2*); 2 threads that execute the flow of undistortion (*und1/2*), detection (*pre1/2*, *inf1/2*, *pos1/2*), tracking (*tck1/2*) and message sending (*msg1/2*); and an optional thread for visualization purposes (*show*). The reason behind this split is to control the end-to-end latency of the application. Having all the steps in sequential order would cause the missing of the 100 ms deadline while having high data age and reaction time latencies. Indeed, the first bottleneck of this application is to retrieve and read the frame from the stream, especially because the minimum resolution of the considered stream in our infrastructure is HD (1920x1080).

The second bottleneck is the complete undistort-detecttrack-send flow. Having these two operations in sequence leads to always working on old data, while separating them improves the performance of the system. To better understand the problem, some experimental results are reported.

The experiments have been carried on an Intel i9-9900KF (@3.60GHz) coupled with an NVIDIA RTX 2080Ti. Two streams were considered: (i) stream [1] with an image resolution of 1920x1080 and rate of 25 FPS, (ii) stream [2] with image resolution 3072x1728 and rate of 30 FPS. A better idea of the timing and the data exchange of the application is depicted in Figure 3.

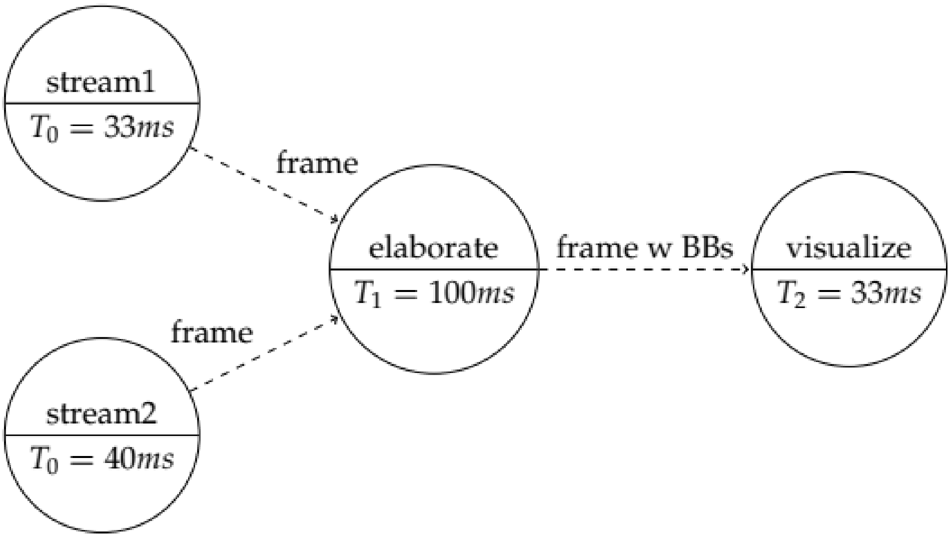


Figure 3 class-edge timing and data exchange details

3.2.2 sensor fusion as a DAG

This application can be seen as a sporadic DAG task with *T* = 100*ms*, and it is depicted as so in Figure 4. The DAG task is both heterogeneous, because multiprocessor CPU and GPU are used, and conditional, given that the visualization is optional. To better understand the flow:

- the subtask *LiDAR* retrieves the point cloud from the sensor, that is then preprocessed by subtask *preL* and then offloaded onto the GPU to permorm the *WBCL* subtask.
- there are four subtask (*cam1* to *cam4*) that collect the frame from the four different cameras. Then the subtask *preC* pre-process them to obtain the format required by YOLO. The pre-processed frames are offloaded onto the GPU and the *inference* is computed. Finally the subtask *postC* applies the post-processing and *project* projects the 2D BBs into the cylindrical representation.
- the subtask *fusion* fuses the output of the clustering and the detection, once the previous subtasks have completed.
- the subtask *show* is in charge of the visualization and it is optional.

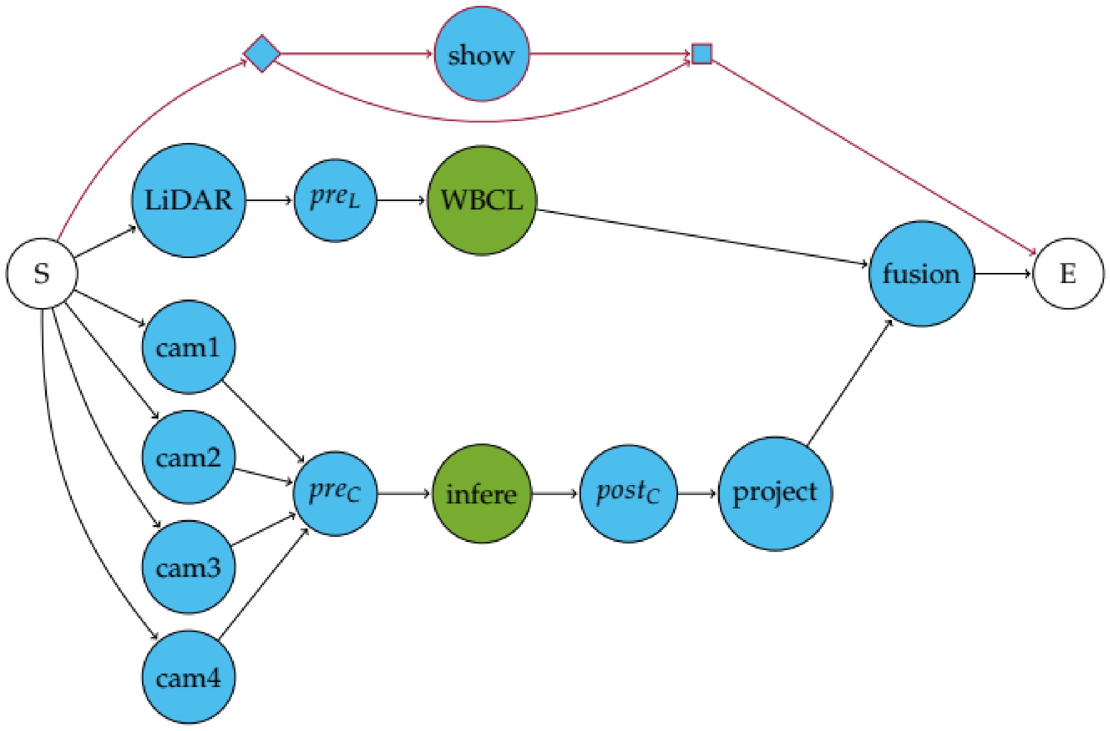


Figure 4 Sensor fusion represented as a Heterogeneous Conditional DAG

This same application could also be modelled as a multi-rate task set, which is depicted in Figure 5. The periods of the sensors are given, *T*0 = 100*ms* for the LiDAR and *T*1 = 33*ms* for the cameras; while the period of the visualization is bounded to *T*5 = 33*ms*. The task *clustering* and *detection* inherit the period from the LiDAR and cameras respectively, but the fusion period needs to be *T*4 = 100*ms* in order to have the data from the LiDAR.

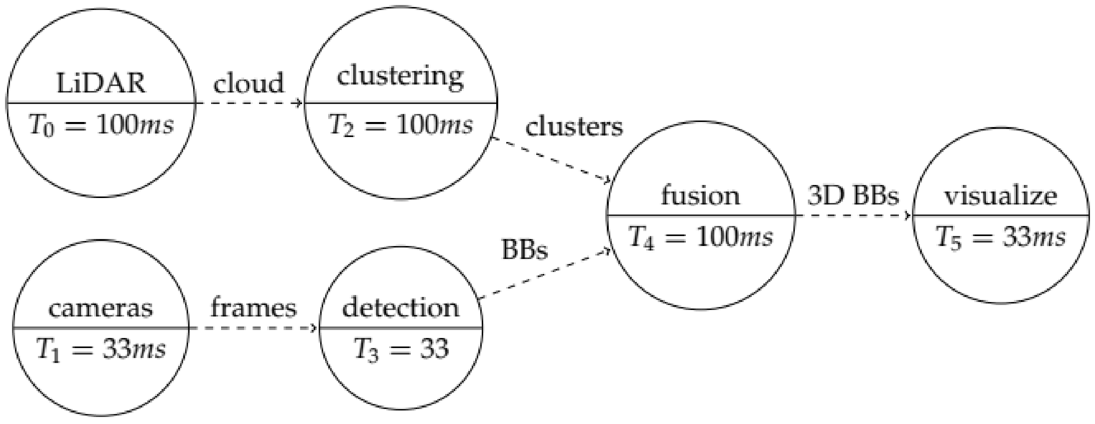


Figure 5 Sensor fusion timing and data exchange details

3.2.3 Deduplication

The deduplication task consist only of one subtask, therefore no DAG equivalent is represented.

3.3 Evaluation Results

3.3.1 Object detection and tracking

Figure 6 reports the minimum, average and maximum latencies of subtask str1|2 over 5k frames, split in three further phases: (i)frame acquisition, form the RTSP stream, (ii) frameresize to a smaller resolution (i.e. 960x540) and frame copy to a shared buffer. From the chart, two main observations can be made.

The first is that the most expensive part is the capture of the stream, which is affected by the original resolution and it's higher for stream [2]. The second is that, even if in average str1|2 take between 15 and 20 milliseconds to execute, the same operation can take up to 160 milliseconds. Given these results, we decided to set 1920x1080 as an upper bound resolution for the RTSP stream, changing the setting directly on the cameras, so that the maximum execution is always less than 100 milliseconds.

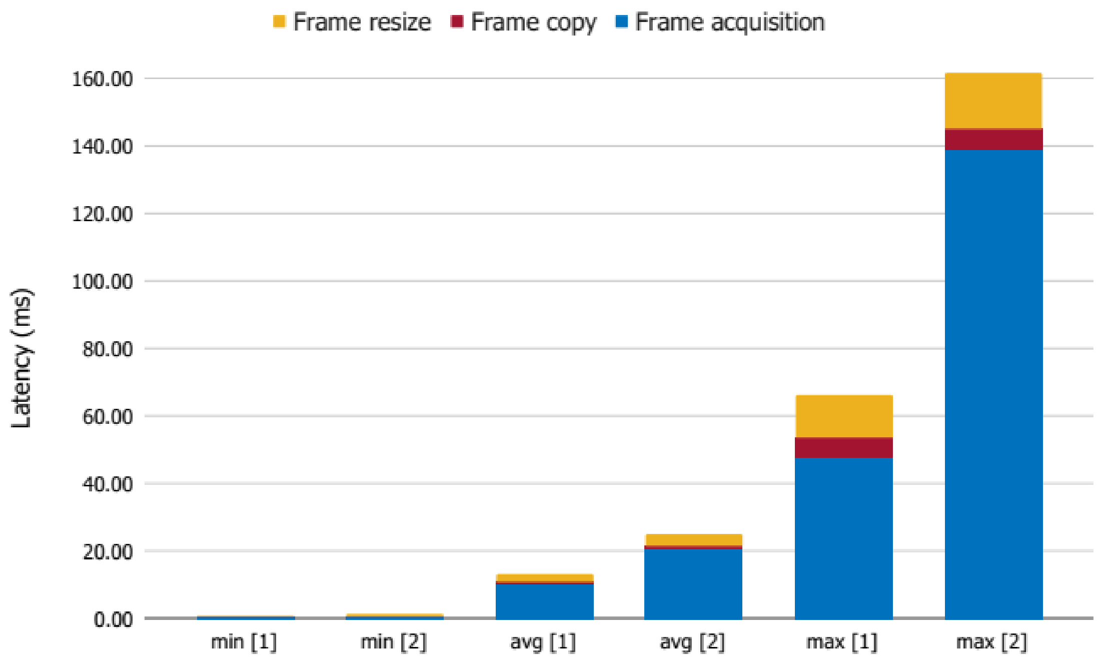


Figure 6 Minimum, average and maximum latencies of subtask str 1|2, split in 3 further phases: frame acquisition, frame resize and frame copy.

A similar chart for the undistort-detect-track-send flow is reported in Figure 7. Additionally to the already described phases, the copy of the frame from the shared buffer and the optional viewer feeding have been profiled. From this chart, we can evince that still there are some differences in the streams, not related to the resolution but on the scene itself instead. Stream [1] comes from a camera that points on a roundabout: it's a dynamic scenario and even though there are many objects, the trackers don't last long. On the other hand, stream [2] comes from a camera that points to a parking lot: it's a static view, there are many objects and their trackers are always alive, keeping their information (with a limited history). For this reason, differences in terms of latency can be found in the tracking and viewer phases, which are proportional to the stored trajectories, while the other phases have similar duration. In any case, the total time of the flow is always less than 100 milliseconds.

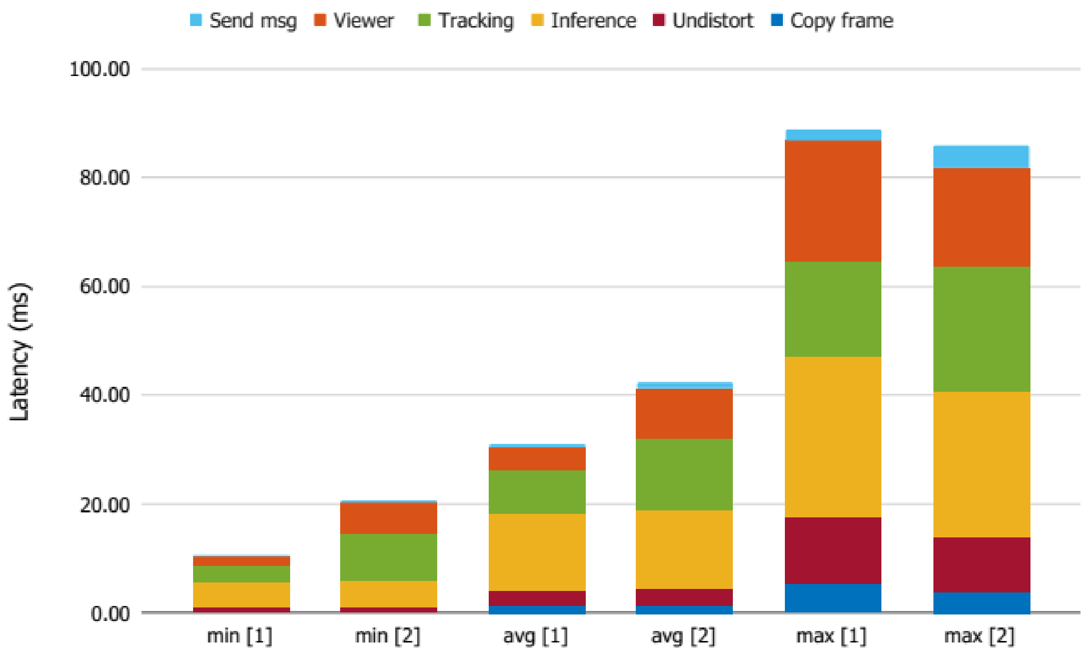
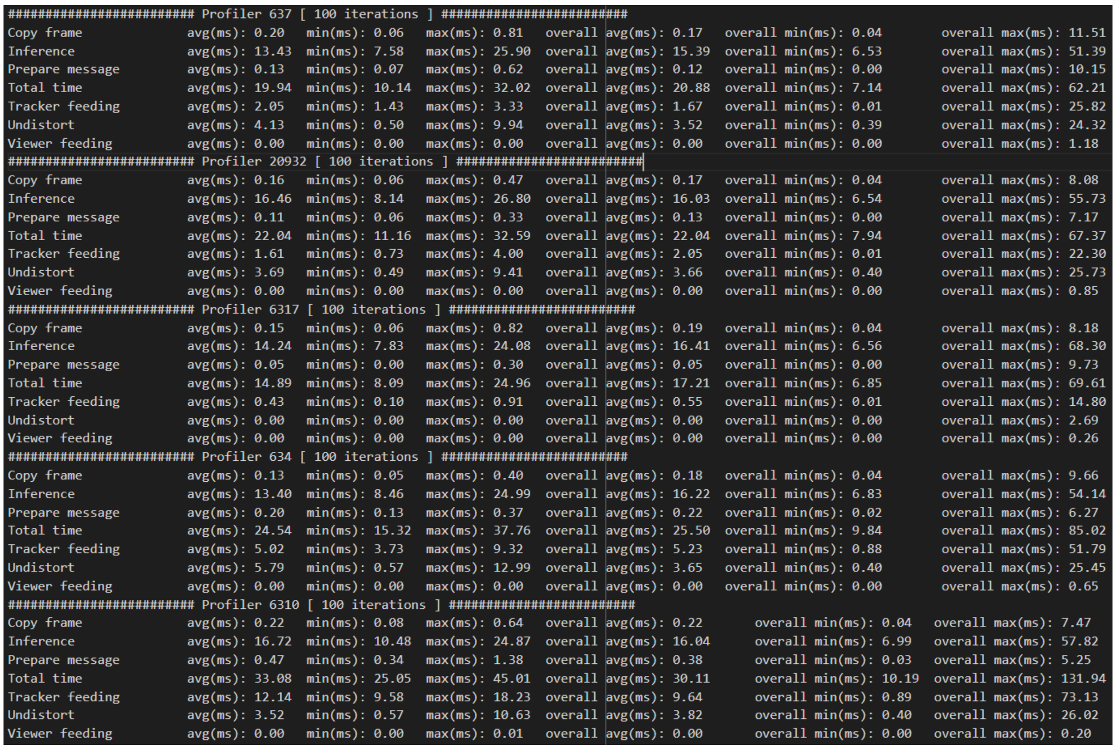


Figure 7 Minimum, average and maximum latencies of undistort-detect-track-send flow.

On a class fog node, we can see the results of the real time elaboration of 100 frames for 5 cameras in Figure 8 as the output of our profiling.

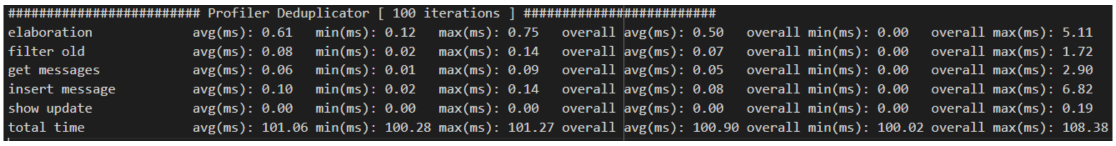


```
######################## Profiler 637 [ 100 iterations ] ########################
Copy frame          avg(ms): 0.20   min(ms): 0.06   max(ms): 0.81   overall avg(ms): 0.17   overall min(ms): 0.04      overall max(ms): 11.51
Inference           avg(ms): 13.43  min(ms): 7.58   max(ms): 25.90  overall avg(ms): 15.39  overall min(ms): 6.53      overall max(ms): 51.39
Prepare message     avg(ms): 0.13   min(ms): 0.07   max(ms): 0.62   overall avg(ms): 0.12   overall min(ms): 0.00      overall max(ms): 10.15
Total time          avg(ms): 19.94  min(ms): 10.14  max(ms): 32.02  overall avg(ms): 20.88  overall min(ms): 7.14      overall max(ms): 62.21
Tracker feeding     avg(ms): 2.05   min(ms): 1.43   max(ms): 3.33   overall avg(ms): 1.67   overall min(ms): 0.01      overall max(ms): 25.82
Undistort           avg(ms): 4.13   min(ms): 0.50   max(ms): 9.94   overall avg(ms): 3.52   overall min(ms): 0.39      overall max(ms): 24.32
Viewer feeding      avg(ms): 0.00   min(ms): 0.00   max(ms): 0.00   overall avg(ms): 0.00   overall min(ms): 0.00      overall max(ms): 1.18
######################## Profiler 20932 [ 100 iterations ] ########################
Copy frame          avg(ms): 0.16   min(ms): 0.06   max(ms): 0.47   overall avg(ms): 0.17   overall min(ms): 0.04      overall max(ms): 8.08
Inference           avg(ms): 16.46  min(ms): 8.14   max(ms): 26.80  overall avg(ms): 16.03  overall min(ms): 6.54      overall max(ms): 55.73
Prepare message     avg(ms): 0.11   min(ms): 0.06   max(ms): 0.33   overall avg(ms): 0.13   overall min(ms): 0.00      overall max(ms): 7.17
Total time          avg(ms): 22.04  min(ms): 11.16  max(ms): 32.59  overall avg(ms): 22.04  overall min(ms): 7.94      overall max(ms): 67.37
Tracker feeding     avg(ms): 1.61   min(ms): 0.73   max(ms): 4.00   overall avg(ms): 2.05   overall min(ms): 0.01      overall max(ms): 22.30
Undistort           avg(ms): 3.69   min(ms): 0.49   max(ms): 9.41   overall avg(ms): 3.66   overall min(ms): 0.40      overall max(ms): 25.73
Viewer feeding      avg(ms): 0.00   min(ms): 0.00   max(ms): 0.00   overall avg(ms): 0.00   overall min(ms): 0.00      overall max(ms): 0.85
######################## Profiler 6317 [ 100 iterations ] ########################
Copy frame          avg(ms): 0.15   min(ms): 0.06   max(ms): 0.82   overall avg(ms): 0.19   overall min(ms): 0.04      overall max(ms): 8.18
Inference           avg(ms): 14.24  min(ms): 7.83   max(ms): 24.08  overall avg(ms): 16.41  overall min(ms): 6.56      overall max(ms): 68.30
Prepare message     avg(ms): 0.05   min(ms): 0.00   max(ms): 0.30   overall avg(ms): 0.05   overall min(ms): 0.00      overall max(ms): 9.73
Total time          avg(ms): 14.89  min(ms): 8.09   max(ms): 24.96  overall avg(ms): 17.21  overall min(ms): 6.85      overall max(ms): 69.61
Tracker feeding     avg(ms): 0.43   min(ms): 0.10   max(ms): 0.91   overall avg(ms): 0.55   overall min(ms): 0.01      overall max(ms): 14.80
Undistort           avg(ms): 0.00   min(ms): 0.00   max(ms): 0.00   overall avg(ms): 0.00   overall min(ms): 0.00      overall max(ms): 2.69
Viewer feeding      avg(ms): 0.00   min(ms): 0.00   max(ms): 0.00   overall avg(ms): 0.00   overall min(ms): 0.00      overall max(ms): 0.26
######################## Profiler 634 [ 100 iterations ] ########################
Copy frame          avg(ms): 0.13   min(ms): 0.05   max(ms): 0.40   overall avg(ms): 0.18   overall min(ms): 0.04      overall max(ms): 9.66
Inference           avg(ms): 13.40  min(ms): 8.46   max(ms): 24.99  overall avg(ms): 16.22  overall min(ms): 6.83      overall max(ms): 54.14
Prepare message     avg(ms): 0.20   min(ms): 0.13   max(ms): 0.37   overall avg(ms): 0.22   overall min(ms): 0.02      overall max(ms): 6.27
Total time          avg(ms): 24.54  min(ms): 15.32  max(ms): 37.76  overall avg(ms): 25.50  overall min(ms): 9.84      overall max(ms): 85.02
Tracker feeding     avg(ms): 5.02   min(ms): 3.73   max(ms): 9.32   overall avg(ms): 5.23   overall min(ms): 0.88      overall max(ms): 51.79
Undistort           avg(ms): 5.79   min(ms): 0.57   max(ms): 12.99  overall avg(ms): 3.65   overall min(ms): 0.40      overall max(ms): 25.45
Viewer feeding      avg(ms): 0.00   min(ms): 0.00   max(ms): 0.00   overall avg(ms): 0.00   overall min(ms): 0.00      overall max(ms): 0.65
######################## Profiler 6310 [ 100 iterations ] ########################
Copy frame          avg(ms): 0.22   min(ms): 0.08   max(ms): 0.64   overall avg(ms): 0.22       overall min(ms): 0.04   overall max(ms): 7.47
Inference           avg(ms): 16.72  min(ms): 10.48  max(ms): 24.87  overall avg(ms): 16.04      overall min(ms): 6.99   overall max(ms): 57.82
Prepare message     avg(ms): 0.47   min(ms): 0.34   max(ms): 1.38   overall avg(ms): 0.38       overall min(ms): 0.03   overall max(ms): 5.25
Total time          avg(ms): 33.08  min(ms): 25.05  max(ms): 45.01  overall avg(ms): 30.11      overall min(ms): 10.19  overall max(ms): 131.94
Tracker feeding     avg(ms): 12.14  min(ms): 9.58   max(ms): 18.23  overall avg(ms): 9.64       overall min(ms): 0.89   overall max(ms): 73.13
Undistort           avg(ms): 3.52   min(ms): 0.57   max(ms): 10.63  overall avg(ms): 3.82       overall min(ms): 0.40   overall max(ms): 26.02
Viewer feeding      avg(ms): 0.00   min(ms): 0.00   max(ms): 0.01   overall avg(ms): 0.00       overall min(ms): 0.00   overall max(ms): 0.20
```

Figure 8 object detection and tracking results

3.3.2 Deduplication

Figure 9 shows the computation time for the deduplicator task on fog node 4, which is the machine that aggregates all the MASA cameras and cars to create a complete snapshot. The times of more than 100 ms are due to a sleep that we introduced in the code to permit our status to be updated in a timely manner. The snapshot in the end is sent to both the Cloud, the smart cars and the municipality visualizer, therefore flooding it with data was creating overhead on the communication media. Therefore, we added the pause in order to obtain a predictable behaviour. However, in the final integration this pause is not needed because the deduplicator task is scheduled by COMPs as a periodic task that can be scheduled with a period of the desired length.



```
######################## Profiler Deduplicator [ 100 iterations ] ########################
elaboration            avg(ms): 0.61   min(ms): 0.12   max(ms): 0.75   overall avg(ms): 0.50   overall min(ms): 0.00   overall max(ms): 5.11
filter old             avg(ms): 0.08   min(ms): 0.02   max(ms): 0.14   overall avg(ms): 0.07   overall min(ms): 0.00   overall max(ms): 1.72
get messages           avg(ms): 0.06   min(ms): 0.01   max(ms): 0.09   overall avg(ms): 0.05   overall min(ms): 0.00   overall max(ms): 2.90
insert message         avg(ms): 0.10   min(ms): 0.02   max(ms): 0.14   overall avg(ms): 0.08   overall min(ms): 0.00   overall max(ms): 6.82
show update            avg(ms): 0.00   min(ms): 0.00   max(ms): 0.00   overall avg(ms): 0.00   overall min(ms): 0.00   overall max(ms): 0.19
total time             avg(ms): 101.06 min(ms): 100.28 max(ms): 101.27 overall avg(ms): 100.90 overall min(ms): 100.02 overall max(ms): 108.38
```

Figure 9 Deduplicator results

3.3.3 Sensor fusion

Table 1 Execution times on the AGX Xavier, statistics computed over 5K frames. reports the results of the tested application on the AGX Xavier in terms of minimum, average and maximum execution time over 5k frames.

	Clustering	Detection	Fusion
min (ms)	1,98	26,07	0,18
avg (ms)	2,96	28,86	0,57
max (ms)	7,44	38,50	1,20

Table 1 Execution times on the AGX Xavier, statistics computed over 5K frames.

4 References

[1] CLASS Consortium, "CLASS: Edge&CLoud Computation: A Highly Distributed Software Architecture for Big Data AnalyticS," Baercelona, 2017.

[2] R. Cavicchioli, *D3.1 Real-Time analysis of the edge computing platform*, 2018.

[3] E. Hadad, "D3.3 Final release Edge Analytics Platform Agent," CLASS, 2020.

[4] R. Cavicchioli, "D3.5 Final release of the realtime analysis methods," CLASS, 2020.

[5] CLASS Consortium, "D1.6 CLASS Use Case Evaluation," 2021.

[6] D. Amendola, "D1.4 - Final Release Of The Smart City Use Case," CLASS, 2020.

[7] A. a. W. C.-Y. a. L. H.-Y. M. Bochkovskiy, "Yolov4: Optimal speed and accuracy of object detection," arXiv preprint arXiv:2004.10934, 2020.

[8] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[9] L. B. F. B. F. G. P. B. a. M. B. M. Verucchi, “Real-Time clustering and LiDAR-camera fusion on embedded platforms for self-driving cars,” in *IEEE Robotic Computing proceedings*, 2020.

[10] J. a. M. S. a. S. R. a. N. A. Elseberg, “Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration,” *Journal of Software Engineering for Robotics (JOSER),* vol. 3, no. 1, pp. 2--12, 2012.