



Design of a Protocol to Enable Spontaneous Platooning

Stefano Fontana, Elena Tonini

Executive Summary

This project aims at designing a protocol to enable spontaneous platooning for communication-enabled vehicles.

The vehicles travel on a three-lane highway at speed in the range between V_{min} and V_{max} , while broadcasting CAMs at 1 Hz. An initiator vehicle may suggest the formation of a platoon to the surrounding vehicles and, once it receives N_p CAMs from another vehicle signaling that they are also willing to form a platoon, it switches CAMs frequency to 10 Hz and starts the platoon formation protocol.

In the initial version of this protocol, we only consider two vehicles as the ones willing to form a platoon: these vehicles must exchange information about their current speed and the desired speed that they would like to travel at once the platoon is formed; moreover, they have to exchange information on their position on the road, to allow a simplification of the platoon leader negotiation phase: the leader of the platoon will by default be identified with the car in front so that no overtaking maneuver needs to take place.

In this report, we provide an analysis of the design and implementation of the protocol and the computational results we obtain from the simulation when varying the controller of the vehicles.

Contents

| | | |
|----------|--|-----------|
| 1 | Project Description | 3 |
| 2 | Protocol Design | 4 |
| 2.1 | Implementation | 4 |
| | Setup of the Simulation | 4 |
| | Code Architecture | 5 |
| | Exchanged Messages Formats | 5 |
| | Finite State Machine | 6 |
| | Additional Vehicles and Merge Maneuver | 8 |
| 3 | Results | 10 |
| 3.1 | Maneuvers per Repetition of the Simulation | 10 |
| 3.2 | Platoon Size by Leader | 11 |
| 3.3 | Maneuver Time Comparison | 12 |
| 3.4 | Number of Maneuvers Comparison | 13 |
| 4 | Conclusions and Future Work | 14 |
| A | Usage Report | 16 |
| A.1 | Build Instructions | 16 |
| | Clone | 16 |
| | Install Dependencies and configure them | 16 |
| | Build | 16 |
| | Run | 16 |
| | Batch Build and Run | 17 |

1 Project Description

The project aims at designing a protocol to enable spontaneous platooning of communication-enabled vehicles.

The initial approach we adopt is to consider all vehicles as communication-enabled, meaning that they could all potentially be willing and able to form platoons. The second assumption that we make is that platoons — at least in the initial phase of the development of the protocol — are only made of two vehicles. This lets us deal with the exchanged messages much more easily and check that the correct behavior is implemented for a basic and close-to-ideal environment before extending the protocol to involve more than two vehicles in the platoon formation, whether that be achieved through a multi-vehicle negotiation of the platoon structure (i.e., who is going to be the leader, in what order the other vehicles should be arranged, etc.) or through a join maneuver.

All vehicles travel along a 10 km-long three-lane highway and by default send CAMs at 1 Hz, announcing their willingness to be part of a platoon driving between V_{min} and V_{max} . These values indicate both the speed limits the vehicles drive in between of when driving “alone” with an ACC model, and the range of acceptable speed that they are willing to travel at when they are part of a platoon.

After hearing N_p CAMs from another vehicle, the platoon formation protocol initiates, requiring the vehicles to switch the CAMs frequency to 10 Hz.

The two vehicles that are involved initially are both considered as leaders of a one-car platoon: this way the platoon formation maneuver can be treated as a “merge platoons” maneuver, which is simpler to manage.

After forming the platoon, the vehicles will continue traveling without changing lanes and the platoon will be open to expansion by either merging with another multi-car platoon or with another one-car platoon.

The relevant aspect in all cases is that all communications and maneuvers are managed by the leaders of the platoons, whether they are single or multi-car platoons. This means that, to form a two-car platoon, the two involved vehicles are both the leader of their own single-car platoon and communicate directly, but, in case we want two multi-car platoons to merge into a longer one, only the leaders of the two platoons will communicate, while all other involved vehicles will simply follow their leader’s instructions.

2 Protocol Design

The protocol was designed incrementally, by subsequently taking into account new aspects to introduce into the environment that we recreate through the simulation. The main aspects of the protocol can be seen in the list below:

1. Definition of a ten-kilometer-long three-lane highway where vehicles equipped with Automatic Cruise Control (ACC) enter at random times and travel keeping their random desired speed V_d . The speed value is taken from the Gaussian distribution;
2. Design of the Finite State Machine (FSM) of the protocol for the generic vehicle (i.e. single-car platoon leader). To simplify the negotiation of the leader of the forming platoon, we require vehicles to be driving on the same lane to allow initiation of the platoon formation. Moreover, we choose as leader the vehicle that is already ahead of the other and as platooning speed the $\min(\max(V_{max1}, V_{max2}))$. Once the platoon forms, the vehicles switch to the Ploeg or CACC controller based on the run configuration.
3. Addition of more vehicles onto the road and introduction of multi-car platoon merge maneuvers;
4. Analysis of the average time it takes the vehicles to form a platoon once the maneuver begins;

A significant (and somewhat basic) hypothesis that we have to consider is that for the vehicles to form platoons it is necessary that, given two vehicles C_1 and C_2 , they will form a platoon when C_1 enters the road before C_2 and $V_{d2} > V_{d1}$; as a matter of fact, if C_1 enters the road before C_2 and drives faster than C_2 , the two vehicles will get further and further away, without having the chance or the time to form a platoon.

2.1 Implementation

Setup of the Simulation

To implement this simulation, we start with the assumption that each vehicle can be treated and managed as a single-vehicle platoon. This way, the action of “two cars forming a platoon” results in a specific version of a “merge platoons” action.

Initially, we want to make sure that the road setup is working by defining a Sumo configuration: we prepare a Highway environment limited to the first lane and then inject two vehicles one after the other with a random injection interval based on a truncated normal distribution with mean and variance of 1 s with a fixed offset of 4 s in order to guarantee safe distancing.

Then we expand the test case with four cars in one single lane, then again with 50 cars in order to check platoon limitation conditions and response with multiple platoons.

Finally, we keep introducing cars for the whole simulation time, getting closer to resembling real traffic. Because of Sumo limitations, we had to constrain platooning on the same lane as the car requesting it. This also means that no lane switching is allowed in our simulation due to the difficulty of implementation, as lane switching has not been efficiently implemented yet in the code base we use as a reference and starting point.

Code Architecture

We developed the simulation environment mimicking Plexe architecture. This resulted in five different components (three of which are management components) and one custom maneuver extended from the standard pool available at the time of development inside the Plexe repository. We wrote a Scenario component responsible for events happening during the simulation. Because no special events like emergency braking should happen inside the simulation, we simply extended the SimpleScenario class.

As for the traffic manager, we needed a continuous injection of traffic, which is implemented inside the class TrafficManager. This class steadily inserts cars as platoons of length 1 with a defined trunc-normal interval, as discussed in the section 2.1.

Every vehicle has a running instance of the application, defined by the class MyPlatooningApp, and a protocol PlatooningProtocol.

The application's inner workings are described in section 2.1: the most relevant detail is that a maneuver request is sent (or the relative ACK is accepted) if and only if the last known position of the platoon in front is reflected by the radar readings. This is because we do not want to start a maneuver when another car is between the two maneuvering parties.

The protocol is responsible for beaconing and communications. It keeps track of the position of the platoons in front and behind and forwards the platooning advertisement beacons to the application only when three packets are received. It also manages the packet header building and sends retries.

Exchanged Messages Formats

This section is an overview of the main fields and features of the messages and packets the vehicles exchange.

PacketHeader

All packets extend this header so that the information it contains is commonly shared.

| | |
|-----------------------|---------------|
| senderAddress long | type short |
|-----------------------|---------------|

PlatoonAdvertiseBeacon

Message sent by any vehicles that suggest creating a platoon. Each vehicle embeds its speed, the lane number, indicating of course which lane the vehicle is currently travelling in, the size of the platoon the vehicle belongs to, the platoon id, the coordinates of the vehicle, and whether it is willing to take part into a platoon.

| | | | | | |
|-------------------------|----------------------|-------------------------|-------------------|------------------|-------------------|
| platoon_speed double | lane unsigned int | platoon_size uint8_t | platoon_id int | coords double | accepting bool |
|-------------------------|----------------------|-------------------------|-------------------|------------------|-------------------|

PlatoonCreateRequest

Message sent by the vehicle that first receives three valid beacons from the same sender; it is used to initialize the communication and suggest platoon formation. The message is sent a maximum of 20 times with a 0.1-second interval until a valid answer is received. This ensures that in case of lost packets, a retry has taken place and the communication was not dropped automatically.

| | | |
|-------------------|------------------|----------------------------|
| platoon_id int | leader_id int | current_position double |
|-------------------|------------------|----------------------------|

PlatoonCreateRequestACK

Message sent by the vehicle that has received a PlatoonCreateRequest message; it is used to confirm the will to form a platoon and allow initiation of the actual platoon formation phase, thus ending the negotiation phase.

| | | | |
|------------------|-----------------|------------------|----------------------------|
| platoonId int | leaderId int | accepted bool | current_position double |
|------------------|-----------------|------------------|----------------------------|

PlatoonAdvertisementListenTimeout

Internal message. Once the count field reaches value 3, it means that our vehicle has received the same PlatoonAdvertiseBeacon $N_p = 3$ times, which is enough to guarantee that the other vehicle is actually interested in forming a platoon and that we have not just overheard a message belonging to a conversation we are not part of.

| | | | |
|-----------------|------------------|--------------------|-------------|
| platoon long | count uint8_t | distance double | lane int |
|-----------------|------------------|--------------------|-------------|

Finite State Machine

To implement the protocol, it is first necessary to outline the behaviour of the involved vehicles using FSM. Knowing that all vehicles initially are treated as leaders of a one-car platoon, they actually all have the same behaviour, which can be described by a single state machine (Fig. 1).

Vehicles start from the APP_LEADER_IDLE state. Upon receiving a PlatoonAdvertiseBeacon, if our vehicle is not the one that is going to be the leader of the two-car platoon and there is some

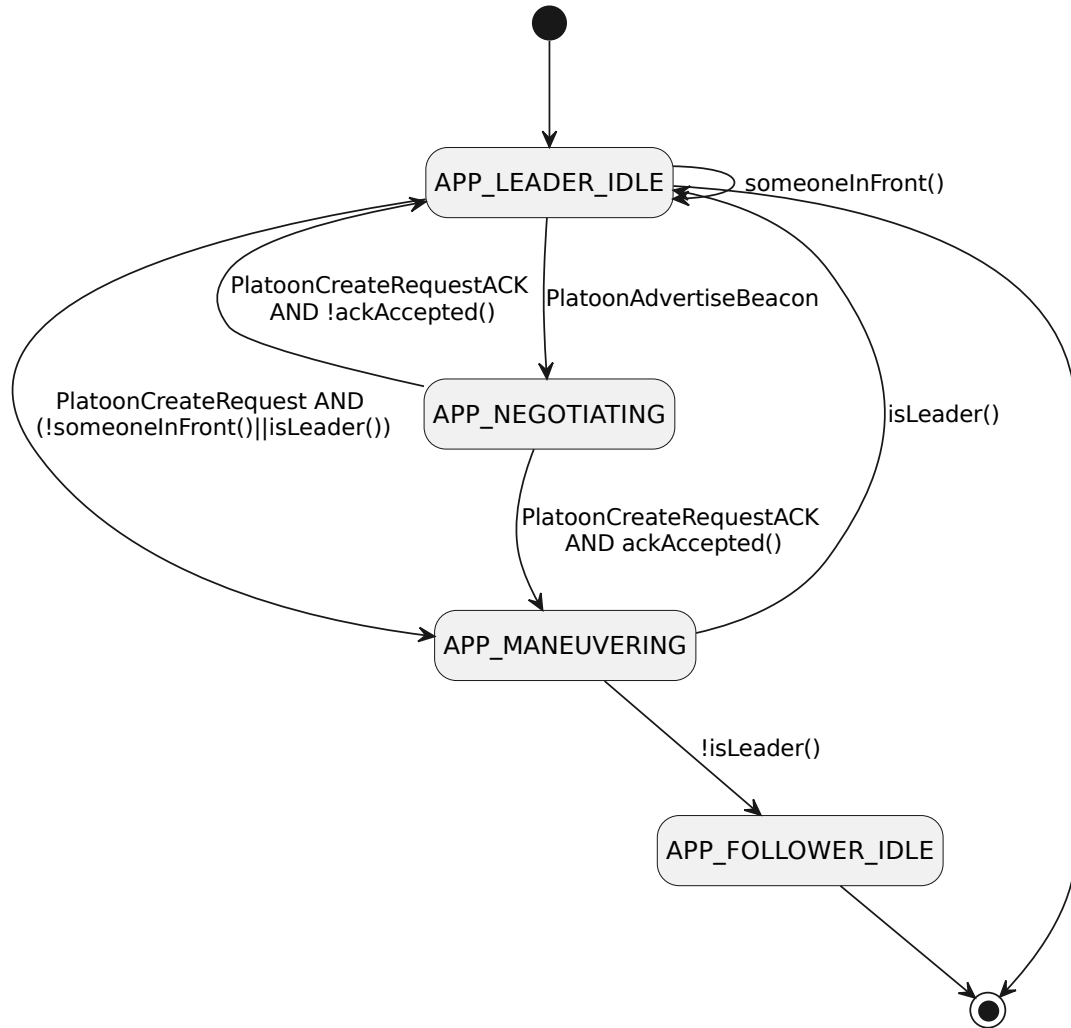


Figure 1: FSM

other vehicle in between us and the future leader of the platoon, the formation is aborted by default because we do not take into account overtaking maneuvers; in such case, the state remains APP_LEADER_IDLE.

If, instead, we are going to be the leader and initiator of the upcoming platoon, we build the PlatoonCreateRequest to reply with and switch to the APP_NEGOTIATING state.

If we are not the initiator of the maneuver, we will receive a PlatoonCreateRequest and switch to APP_MANEUVERING state (provided that either we are the leader or that there is no one between us and the leader).

In case we are in APP_NEGOTIATING state and the PlatoonCreateRequest receives an ACK response refusing the platoon formation, we revert back to the APP_LEADER_IDLE state, whereas if we receive an ACK accepting the platoon formation request we switch to APP_MANEUVERING state. From this point on, the only change of state we can undergo depends on whether we are the leader or the follower in the two-cars platoon: in the first case, we switch to APP_LEADER_IDLE, in the

second case we switch to APP_FOLLOWER_IDLE.

The maneuver has now ended and the leader of the newly formed two-car platoon can now start new maneuvers with other vehicles that may want to join the platoon or merge their multi-car platoon with this one.

Additional Vehicles and Merge Maneuver

The simulation was initially performed having the traffic manager only introduce two vehicles on the same lane of the road, but as soon as the protocol was working on two vehicles we introduced additional ones, making sure that the protocol would be working with more realistic traffic as well.

The vehicles are added onto the highway and they are ACC-driven by default. Every additional vehicle is also associated with a platoon id, given that, as was previously said, every vehicle is nothing but a one-car platoon.

The behavior of the simulation with more than two vehicles is a simple expansion of the original behavior: once two vehicles end the initial platoon formation that is allowed by the initial version of the protocol, the leader of the platoon can start communicating with other platoon leaders and apply the exact same protocol as they already did to potentially merge their platoons into a larger one. This can easily be done because no significant modification to the code base of the simulation is required: by treating vehicles as platoons ever since the beginning, we do not need any additional line of code to manage platoons themselves; moreover, the “merge platoons” maneuver is already implemented in Plexe, therefore we can use it directly to manage multi-car platoons interactions.

The following configurations have been used to obtain the results we will discuss later:

```
1 Config SimpleTwoCars]
2 sim-time-limit = 1000 s
3 repeat=10
4 *.node[*].prot.platooningFormationSpeedRange = 0.5
5 *.node[*].prot.maxPlatoonSize = 8
6 *.manager.command = "sumo"
7 *.node[*].scenario_type = "Scenario"
8 **.traffic.nCars = 2
9 **.traffic.nLanes = 1
10 **.traffic.insertStartTime = 0.1s
11 **.traffic.platoonInsertSpeed = uniform(15mps, 20mps)
12 **.traffic.insertDelay = truncnormal(1s, 1s)
13 *.node[*].scenario.targetController = "PLOG"
14 output-vector-file = ${resultdir}/${configname}_${repetition}.vec
15 output-scalar-file = ${resultdir}/${configname}_${repetition}.sca
16 *.node[*].maneuverSpeed.result-recording-modes = +histogram
17
18 [Config SimpleFourCars]
19 extends = SimpleTwoCars
20 **.traffic.nCars = 4
21 *.node[*].prot.maxPlatoonSize = 8
22
23 [Config SimpleHighNumberPLOG]
24 extends = SimpleTwoCars
25 **.traffic.nCars = 1000
26 sim-time-limit = 1000 s
27 *.node[*].prot.maxPlatoonSize = 16
```

```
28 *.node[*].scenario.targetController = "PLOEG"
29 **.traffic.insertDelay = truncnormal(1s, 1s)
30
31 [Config SimpleHighNumberCACC]
32 extends = SimpleTwoCars
33 **.traffic.nCars = 1000
34 *.node[*].prot.maxPlatoonSize = 16
35 *.node[*].scenario.targetController = "CACC"
36 **.traffic.platoonLeaderHeadway = 3s
37 **.headway = 2s
38 *.node[*].scenario.accHeadway = 2s
39 *.node[*].scenario.leaderHeadway = 2s
```

The first configuration was used to produce the results relative to the two-cars-only simulation, while the others — as can be seen — simply extend the initial configuration and increase the number of vehicles and/or lanes. It is important to notice that we have set the maximum platoon size to 15 vehicles and that the speed at which the vehicle starts traveling is randomly drawn from a uniform distribution taking values between 35 and 37 meters per second (roughly between 126 and 133 km/h).

While the first simulation only accounts for two cars traveling over the only available lane, the configuration named `HighDensityTraffic` consists of 250 cars: this final configuration is the one that we use to produce experimental data.

3 Results

The analysis we carried out in this study revolves around two controllers whose performances we have chosen to compare: Ploeg and Cooperative Adaptive Cruise Control (CACC) as was implemented in Plexe. Ploeg is inspired by “standard” ACC, as it uses only information from the vehicle in front to manage acceleration and distance between vehicles, also exploiting the desired acceleration of the preceding vehicle. Its similarity with the well-known ACC makes it easier to introduce into the market, especially due to the “social acceptance” of the public towards a controller that behaves more like a human driver compared to CACC, which *per se* results in slightly more sudden maneuvers.

3.1 Maneuvers per Repetition of the Simulation

Changing the seed of the simulation repetition by repetition, we can obtain a significant comparison of the behavior of the two controllers.

In Figure 2b, we can observe that the number of maneuvers the CACC-controlled vehicles take part in is notably lower than the corresponding value for the Ploeg controller (displayed in Figure 2a) except for the third simulation run. This difference is due to the fact that the CACC controller tends to be less effective in preventing car crashes, therefore the number of maneuvers initiated by the vehicles is lower than Ploeg’s because the simulations with CACC automatically stop much sooner due to car accidents, which are instead often avoided using Ploeg controller.

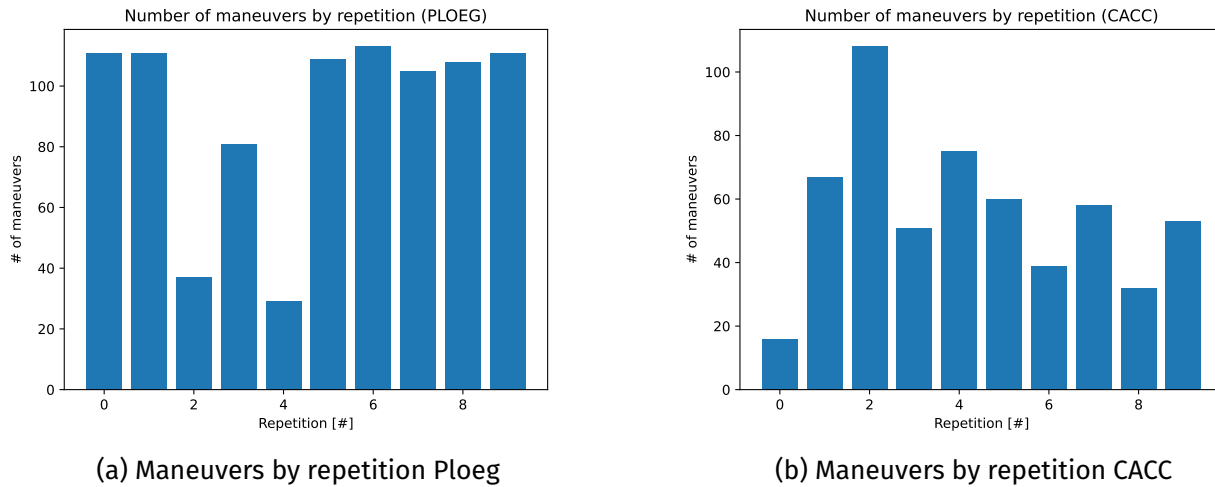


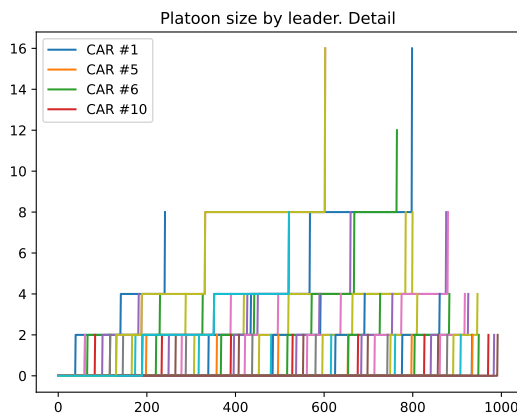
Figure 2: Maneuvers vs Repetitions, comparison

3.2 Platoon Size by Leader

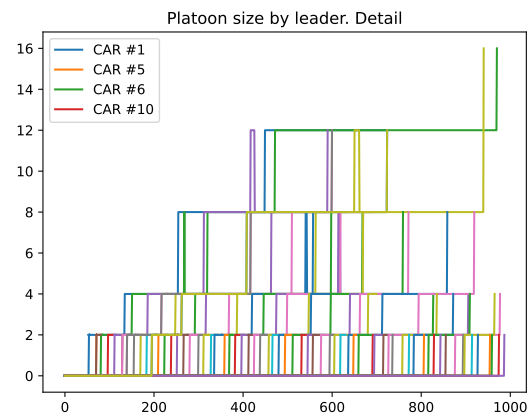
We analyzed the data relative to the platoon size in both CACC and Ploeg scenarios. In both cases, as seen in Figure 3, we see a lot of leaders having the lead of a two-car platoon. Those small platoons persist throughout the whole run because of the traffic injection during the whole simulation's time.

This means that a lot of small platoons form at the beginning of the road, then those platoons merge going forward, resulting in a few big platoons.

We can observe that almost all the traces stop before the simulation time ends: this is because the platoon with that leader id has merged in another platoon and the leader has become a follower of the platoon in front. We can state that the trace stops at the last merge maneuver for a given leader car.



(a) Platoon size by leader, Ploeg, time series



(b) Platoon size by leader, CACC, time series

Figure 3: Platoon size by leader, time series

We can then use the figure 4 in order to make sense of the process: the figure shows cars 12 to 20 during their journey.

- At time 130 s Car #12 joins a platoon of size 2 with Car #13. Note: Car #13, #15, #17, #19, and #21 (the followers of each platoon) are not shown.
- At time 150 s Car #14 joins a platoon of size 2 with Car #15.
- At time 160 s Car #16 joins a platoon of size 2 with Car #17.
- At time 170 s Car #18 joins a platoon of size 2 with Car #19.
- Shortly after, at time 175 s Car #12 and its follower join a platoon of size 4 with Car #14 and its follower.
- At time 185 s Car #20 joins a platoon of size 2 with Car #21.
- At time 270 s Car #16 and its follower join a platoon of size 4 with Car #18 and its follower.

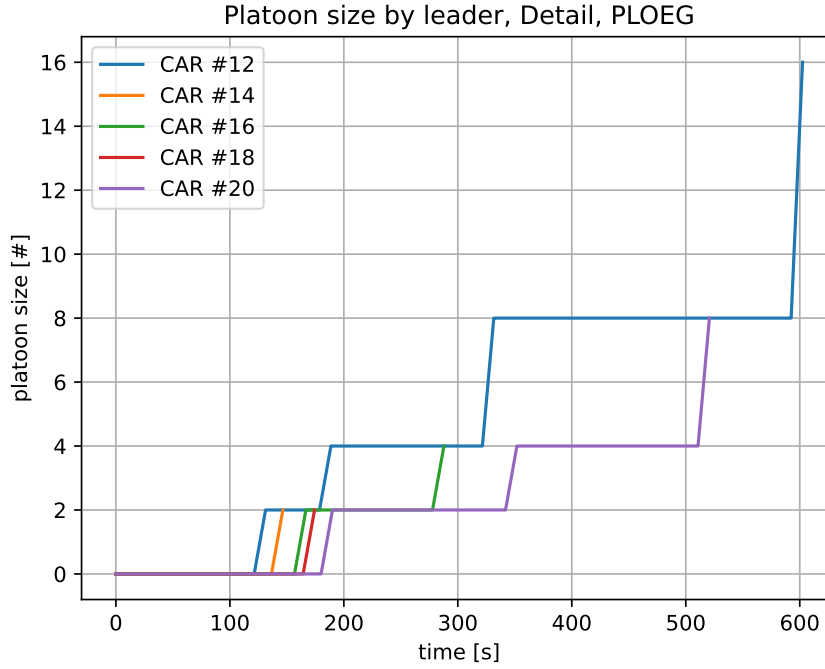


Figure 4: Platoon size by leader, Ploeg, time series, detail

- At time 320 s Car #12 and its followers join a platoon of size 8 with Car #16 and its followers.
- With a short delay, at time 350 s Car #20 and its follower join a platoon of size 4 with a platoon of size 2 behind not shown in the graph.
- With a short delay, at time 510 s Car #20 and its followers join a platoon of size 8 with a platoon of size 4 behind not shown in the graph.
- Lastly, Car #12 and its followers join a platoon of size 16 with Car #20 and its followers at time 595 s.

3.3 Maneuver Time Comparison

We then proceeded with the evaluation of the two controllers. First, we have created a box plot representing the maneuver time for Ploeg and CACC. We chose to use the following metric:

$$M(run, controller) = \frac{\max(\text{maneuvers}(run, controller).time)}{|\text{maneuvers}(run, controller)|}$$

We chose it because it is a valid metric for maneuver frequency.

As we can see from figure 5 Ploeg does allow for more maneuvers in the given simulation time with respect to CACC. This is supposed to happen because of the more continuity Ploeg gives to the

traffic limiting the amount and entity of accelerations and decelerations.

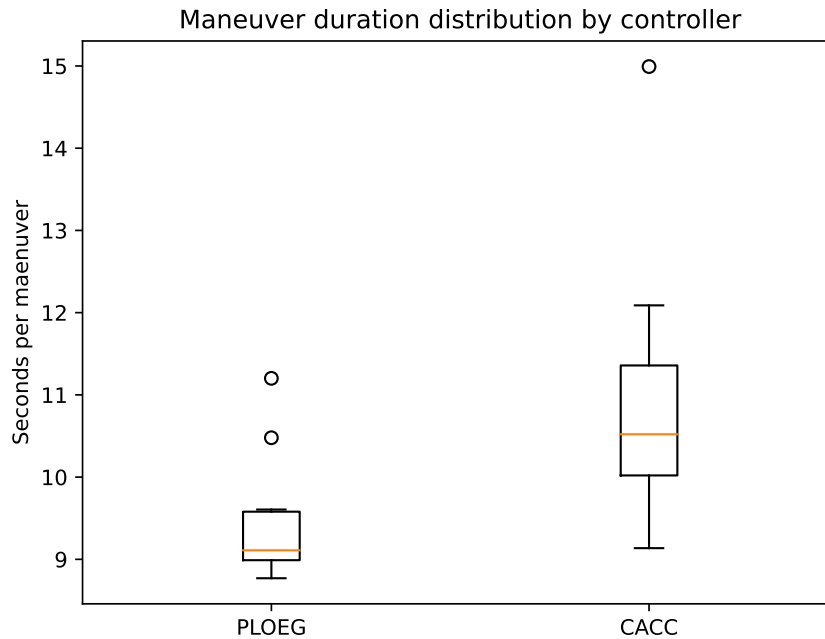


Figure 5: Maneuver time comparison

3.4 Number of Maneuvers Comparison

We concluded the analysis by looking at the raw number of maneuvers made into any simulation run. If in the previous section, Section 3.3, simulation time didn't have an impact because we chose to analyze the required time for the single maneuver, in this result we must consider it. Because of the poor string stability of CACC almost any run ended with an accident. This causes the drop in the number of maneuvers — as seen in figure 2b — because the available time before an accident took place was less.

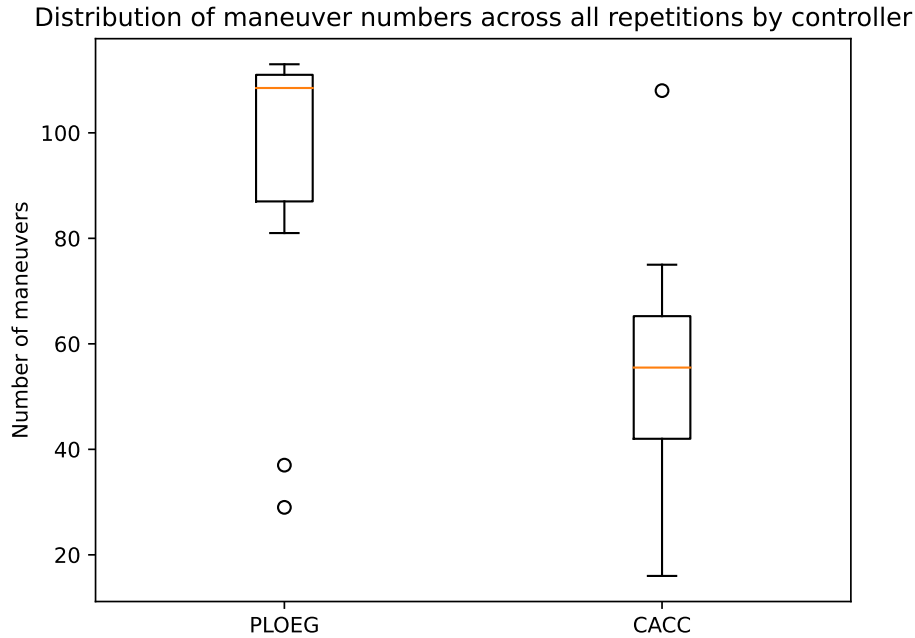


Figure 6: Number of maneuvers comparison


4 Conclusions and Future Work

As we have discussed in this report, our implementation of the protocol for spontaneous platooning only allows platooning with vehicles already traveling on the same lane. Essentially, vehicles can travel in a more compact way and drivers can spend less energy and resources than what would be required by a fully human-driven traffic, which instead would imply unexpected accelerations or decelerations that increase fuel consumption and need higher attention levels from the drivers.

An extension to the version of the protocol we suggested may be to introduce lane changes to make traffic more realistic by exploiting an already-implemented maneuver. Platoon lane changes were first introduced in Plexe 2.1 as an experimental feature and it would be ideal to use them for both single cars and multi-car platoons changing lanes; as we have said multiple times in the report, given our implementation of single cars as leaders of a one-car platoon, it would be much easier to manage the introduction of lane changes for both single vehicles and “traditional” platoons, as the maneuver would need to be introduced only once and it would already work for both cases.

Another aspect that could be interesting to analyze is the protocol’s behavior in the presence of a multitude of human-driven vehicles. It would be relevant to know if the average time it takes to form platoons remains unaltered even when more human-driven traffic is injected into the highway, as well as how traffic interferes with the communication-enabled vehicles.

For instance, are platoons still as long with more human-driven vehicles as with communicat-



ing vehicles? Do platoons still form as frequently? Do platoons overtake human-driven vehicles to merge with other platoons or do they tend to stay in the same lane, given the difficulties of managing multi-car lane switching? More generally, how does human-driven traffic interfere with the implementation of our protocol?

The analysis we illustrated in this report consists only of a partial representation of the overall scenario that we can deal with in the real world but, having to come to terms with the representative power of a model, it gives back an accurate description of how an implementation of the Spontaneous Platooning Protocol could work and the main features it would be characterized by.

A Usage Report

A.1 Build Instructions

Clone

Clone the repository with all its submodules:

```
1 git clone git@github.com:tetofonta/vehicular-networks-and-cooperative-driving.git --recursive
```

Install Dependencies and configure them

Then you should install all the required dependencies:

- cmake minimum version 3.22:

```
1 apt install cmake
```

- omnetpp minimum version 6.0, and it should be inserted into the path variable:

```
1 PATH="$PATH:/path/to/omnetpp/bin"
```

- veins minimum version 5.2, and it should be inserted into the path variable:

```
1 PATH="$PATH:/path/to/veins/bin"
```

- plexe (michele-segata/plexo) minimum version 3.1, and it should be inserted into the path variable:

```
1 PATH="$PATH:/path/to/plexo/bin"
```

Build

```
1 cd vehicular-networks-and-cooperative-driving
2 cmake -B "$PWD/build" -S "$PWD" -DCMAKE_BUILD_TYPE=Debug
3 cmake --build "${PWD}/build" --target libvncd -j $(nproc)
```

Run

```
1 cmake --build "${PWD}/build" --target run_vncd
```


Batch Build and Run

```
1 cd vehicular-networks-and-cooperative-driving
2 cmake -B "${PWD}/build" -S "$PWD" -DCMAKE_BUILD_TYPE=Release
3 cmake --build "${PWD}/build" --target libvncd -j $(nproc)
4 cmake --build "${PWD}/build" --target runmaker
5
6 runmaker4.py "${PWD}/build/runs.txt" -j $(nproc)
```

CopyRight



This work is licensed by the authors under the license Creative Commons 4.0 CC Attribution-NonCommercial-ShareAlike (<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>).

You can reuse and share the material also for derivative work within the limits allowed by the license and with the proper attribution.