MISSIONE 4
ISTRUZIONE
RICERCA

FORESEEN

Deliverable D3.1: Report on how to include and configure attacks in co-simulation architectures

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
Università di Pisa

MISSIONE 4
ISTRUZIONE
RICERCA

FORESEEN

FORmal m**E**thod**S** for attack d**E**t**E**ction in autonomous drivi**N**g systems

PRIN 2022 PNRR

Project number: P2022WYAEW
CUP: I53D23006130001

Deliverable D3.1: **Report on how to include and configure attacks in co-simulation architectures**

Project Start Date: 30/11/2023 **Duration**: 24 months

Coordinator: *University of Pisa*

Deliverable No	D3.1
WP No:	WP2
WP Leader:	RU-MI
Tasks:	T2.3 - Leader: RU-MI T2.4 - Leader: RU-PI
Due date:	M9-12
Delivery date:	November 30, 2024
Authors:	RU-MI, RU-PA, RU-PI, RU-MOL

Dissemination Level:

PU	Public	X
PP	Restricted to other program participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contents

List of Acronyms

CACC	Cooperative Adaptive Cruise Control
CAN	Controller Area Network
DSRC	Dedicated Short-Range Communications
V2V	Vehicle to Vehicle
V2N	Vehicle to Network

1. Introduction

This deliverable shows the extension of the model of vehicles (defined in WP2 – deliverable D2.1) to implement attack injection on the Platoon, starting from attack scenarios (defined in WP1, deliverable D1.3). Attack injection is implemented by modeling the possibility of activating the attack upon the verification of certain conditions (e.g., the starting time) and including the effect of the attack in the model of the vehicle.

Moreover, this deliverable will provide information regarding the data generation and collection for successive analysis.

2. Modelling attack effects in the system

As it has been mentioned in Deliverable D1.3 "Report on attack scenarios and their impact on the sensory and actuation systems", we decided to consider two types of attacks, namely ***internal*** and ***external***. Each attack type can have an impact on two different components of the vehicle, the ***actuator*** and the ***sensors***. The "class" of attacks that we decided to take into account are data alteration attacks.

To properly understand our decision making process during the attack injection phase, we need to recall the multi-model's structure in terms of input and output connections. The set of FMUs present in the Vehicle to Edge Network (V2E) configuration are:

- *Leader* FMU: it will send its position, speed and acceleration to the network; it doesn't have any input.
- *Follower Car's Dynamics* FMU: it will send its position, speed and acceleration to the network and receives the desired acceleration from the network.
- *V2E Network FMU*: this FMU will take every set of position, speed and acceleration values from the leader and from each follower car. Within the network FMU there is the control algorithm, thus it will calculate the desired acceleration for each follower car and simply output it to the vehicles.

The set of FMUs present in the Vehicle to Vehicle (V2V) configuration are:

- *Leader* FMU: it will send its position, speed and acceleration to the network; it doesn't have any input.
- *CACC algorithm FMU* : one instance of such FMU per each vehicle X.
  The CACC FMU associated to vehicle X, will require
  - the set of acceleration, speed and position of vehicle X,
  - the set of acceleration, speed and position of the preceding vehicle of X and
  - the set of acceleration, speed and position of the Leader

  This FMU will give as output
  - the desired acceleration for vehicle X.
- *Follower Car's Dynamics* FMU: it will send its position, speed and acceleration to the network and receives the desired acceleration from the network.

- *V2V Network FMU*: this FMU will simply interconnect the position, speed and acceleration sent by every car X to CACC FMU of vehicles, accordingly.

In both cases, the *Follower Car's Dynamics* FMU will take the acceleration from a component and output to somewhere else the data read from its sensors.

For what concerns the effects of the attack, we could distinguish just two classes: ***Sensor*** attacks and ***Actuator*** attacks.

In fact, in case of a *Sensor attack* between vehicle *j* and *j+1*, it doesn't matter if the data alteration is done before outputting the data from vehicle *j* (Internal attack), inside the network (external attack), or after the input in vehicle *j+1*'s CACC (Internal attack). The effect will be the same in any case, for this reason we simply decided to model it just once. For simplicity, it is modeled before outputting the values of the position, speed and acceleration.

Same goes for the *Actuator attacks*, which instead are being modeled after the input of the desired acceleration, taken from the vehicle's CACC or from the MEC.

An attack can be dormient for some simulated time, and then it starts. The start time is defined in a parameter called *attack_time*.

In the following subsections we will take a look at the different type of attacks per each class.

2.1 Sensor Attacks

We decided to model three different type of attacks to sensors:

- Data alteration with a *low frequency Sine function*
- Data alteration with a *high frequency Sine function*
- Data alteration with a *combination* of both high and low frequency Sine functions.

In order to be able to model different attack effects, it is possible to modify the frequency and amplitude of the sine functions, in addition to the start time of the attack. This can be done by simply adding a few parameters to the *Follower Car's Dynamics* FMU.

In order to make the attack as subtle as possible and harder to detect, we decided to "modify in a coherent way every data sent from the vehicle". The data alteration following one of the three possibilities is done to the acceleration value to be sent outside, then the resulting value is being integrated accordingly, in order to obtain a coherent modification of the speed. Then, the new speed value is integrated again to obtain the position. We have the tuple (acc, speed, position).

The Sensor type of attacks can be done both on a *generic vehicle* in the platoon but also on the *leader*. The Actuator type of attacks can only be done to the *generic vehicle*, but not to the leader. Both attacks require the specification of the time at which the attack starts.

2.2 Actuator Attacks

Here the idea is to modify the value of the desired acceleration taken by the control algorithm before providing it to the actuator. The alteration can be the following:

- altered_value = desired_acceleration * scale
- altered_value = desired_acceleration + constant_factor

Once again, the FMU will require new parameters for the scale or the constant_factor, as well as the time at which the attack starts.

3. Extension of the models to implement attack injection

Only one attack is activated at a time. The injection of the attack is modeled with a parameter of the co-simulation named `attacked`.

The Lead car and the followers are modeled in Simulink [MathWork]. Simulink allows to design models using the graphical tool by adding and interconnecting blocks, which can be input, output, Matlab function and many other types of blocks.

Figure 1 shows the Simulink model of the Leader, with attacks. The Lead car model is at the bottom on the left side of the figure; the attacks model is at the top on the right side of the figure.

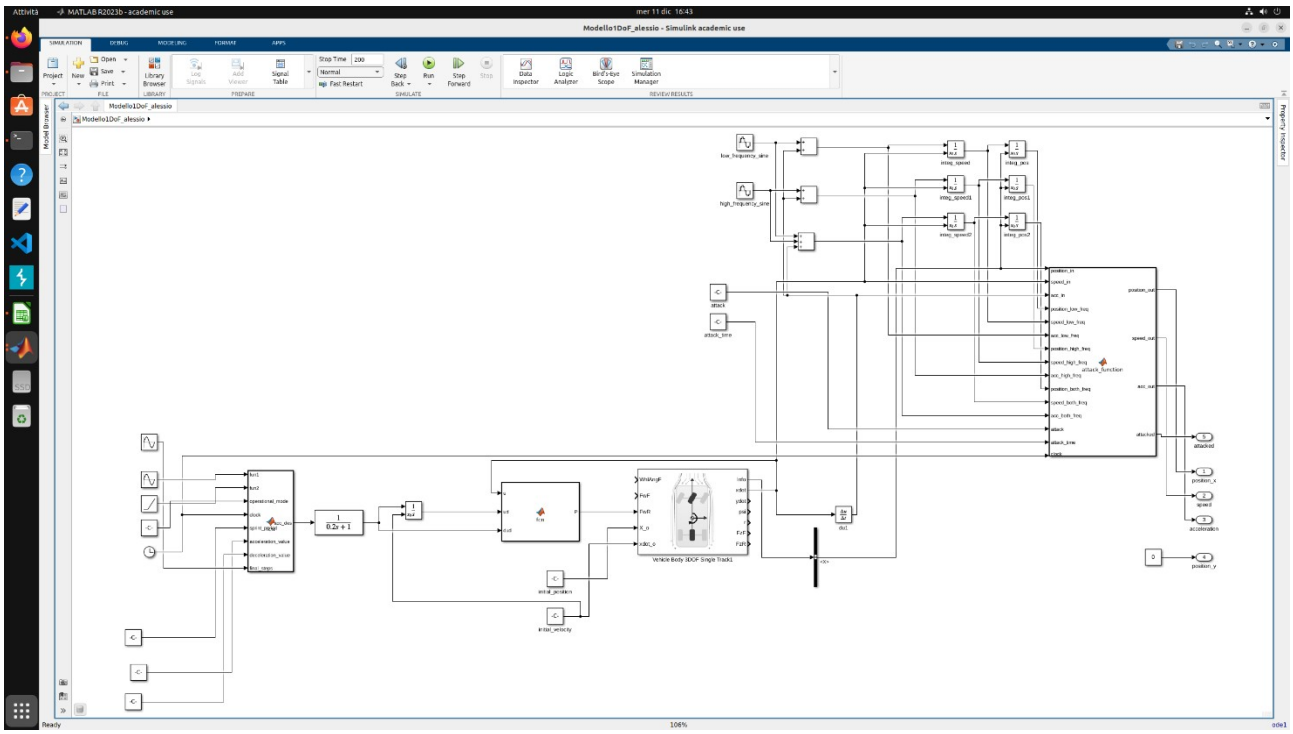

Figure 1: Simulink model of the Leader with attacks

Similarly, Figure 2 shows the Simulink model of the follower (generic car) with attacks. The generic car model is at the bottom on the left side of the figure, the attack models are at the top on the right side of the figure.

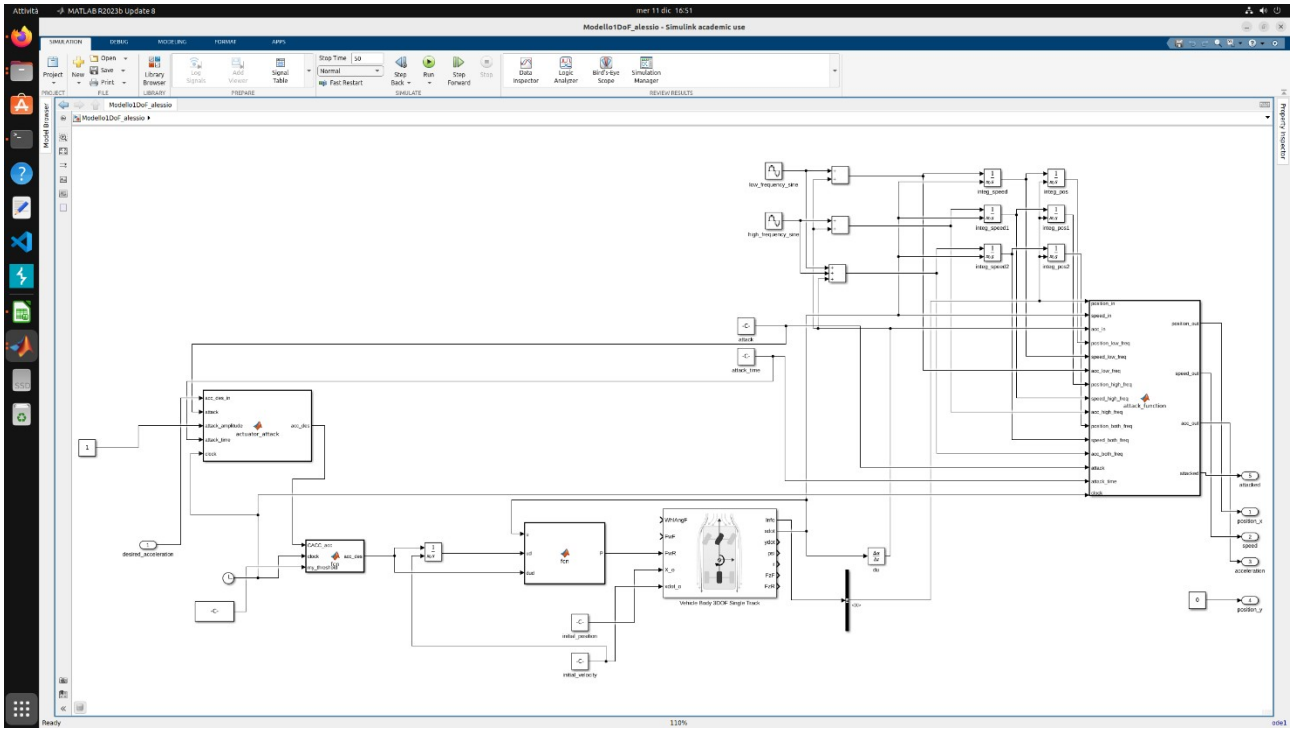

Figure 2: Simulink model of the follower vehicle with attacks

4. Attack functions

The following function (attack_function) is implemented to inject attacks for both the Leader and the follower vehicle [Ber2020].

```
function [position_out, speed_out, acc_out, attacked] = attack_function(position_in, speed_in, acc_in, position_low_freq, speed_low_freq, acc_low_freq, position_high_freq, speed_high_freq, acc_high_freq, ... position_both_freq, speed_both_freq, acc_both_freq, attack, attack_time, clock)


 position_out = position_in;
    speed_out = speed_in;
    acc_out = acc_in;
    attacked = 0;
    if attack_time <= clock
```

```
        if attack != 0
        switch attack
case 1
                    % low frequency sine
                    position_out = position_low_freq;
                    speed_out = speed_low_freq;
                    acc_out = acc_low_freq;
                    attacked = 1;
                case 2
                    % high frequency sine
                    position_out = position_high_freq;
                    speed_out = speed_high_freq;
                    acc_out = acc_high_freq;
                    attacked = 1;
                case 3
                    %both
                    position_out = position_both_freq;
                    speed_out = speed_both_freq;
                    acc_out = acc_both_freq;
                    attacked = 1;
                    %case 4 and 5 are on the actuator
            end
        end
    end
end
```

The following function (actuator_attack) is implemented to inject attacks for follower vehicles.

```
function acc_des = actuator_attack(acc_des_in, attack, attack_amplitude,
attack_time, clock)
    acc_des = acc_des_in;
    if attack_time <= clock
        switch attack
            case 4
                acc_des = acc_des_in * (1 + attack_amplitude);
            case 5
                acc_des = acc_des_in + attack_amplitude;
        end
    end
end
```

5. DSE configuration

Here is the json of the configuration file used by the Design Space Exploration (DSE) tool [Gam2017] to run a batch of co-simulations.

```
{
  "algorithm": {
    "type": "exhaustive"
  },
  "objectiveConstraints": [],
  "objectiveDefinitions": {
    "externalScripts": {},
    "internalFunctions": {}
  },
  "parameterConstraints": [],
  "parameters": {
    "{Leader}.LeaderInstance.initial_position": 0,
    "{Leader}.LeaderInstance.initial_velocity": 0,
     "{Leader}.LeaderInstance.operational_mode": [ 0,1, 2 ],
    "{Network}.NetworkInstance.platoon_0_0_length": 4,
    "{Network}.NetworkInstance.platoon_0_1_length": 4,
```

```
    "{Network}.NetworkInstance.platoon_0_2_length": 4,
    "{Network}.NetworkInstance.platoon_size": 10,
    "{Network}.NetworkInstance.platoon_0_3_length": 4,
    "{Network}.NetworkInstance.platoon_0_4_length": 4,
    "{Network}.NetworkInstance.platoon_0_5_length": 4,
    "{Network}.NetworkInstance.platoon_0_6_length": 4,
    "{Network}.NetworkInstance.platoon_0_7_length": 4,
    "{Network}.NetworkInstance.platoon_0_8_length": 4,
    "{Network}.NetworkInstance.platoon_0_9_length": 4,
    "{Network}.NetworkInstance.network_downlink_delay": [ 0,0.1 ],
    "{Network}.NetworkInstance.network_uplink_delay": [ 0,0.1 ],
    "{Car2}.CarInstance_2.initial_position": 0,
    "{Car2}.CarInstance_2.initial_velocity": 0,
    "{Car2}.CarInstance_2.vehicle_starting_time": 8,
    "{Car2}.CarInstance_2.attack": [ 0, 1,2,3,4,5 ],
    "{Car2}.CarInstance_2.attack_amplitude": 0.2,
    "{Car2}.CarInstance_2.attack_time": 30,
    "{Car2}.CarInstance_2.high_frequency": 172,
    "{Car2}.CarInstance_2.low_frequency": 0.1
  },
  "ranking": {
    "pareto": {}
  },
  "scenarios": []
}
```


6. Testing the attacks in the system

The testing phase is necessary, both to ensure that the attacks behave as expected and also to define a set of interesting values to be used for the data gathering process. For example, a very low value for the amplitude of the low frequency sine function on the sensor attack type will probably have little to no impact on the system, thus making it not interesting, but also an excessive value will of course cause harm, but also be very easily detectable.

Here we will provide a few images of the graphs obtained during said tuning and testing phase, highlighting the effects of the attacks in the system.

Case 1. We introduce an attack where the Car2 outputs wrong data to the Edge; the type of alteration is the low frequency sine function. Figure 3 shows the position of vehicles. Figure 4 shows the acceleration of the Leader and the acceleration of Car 2 (attacked).

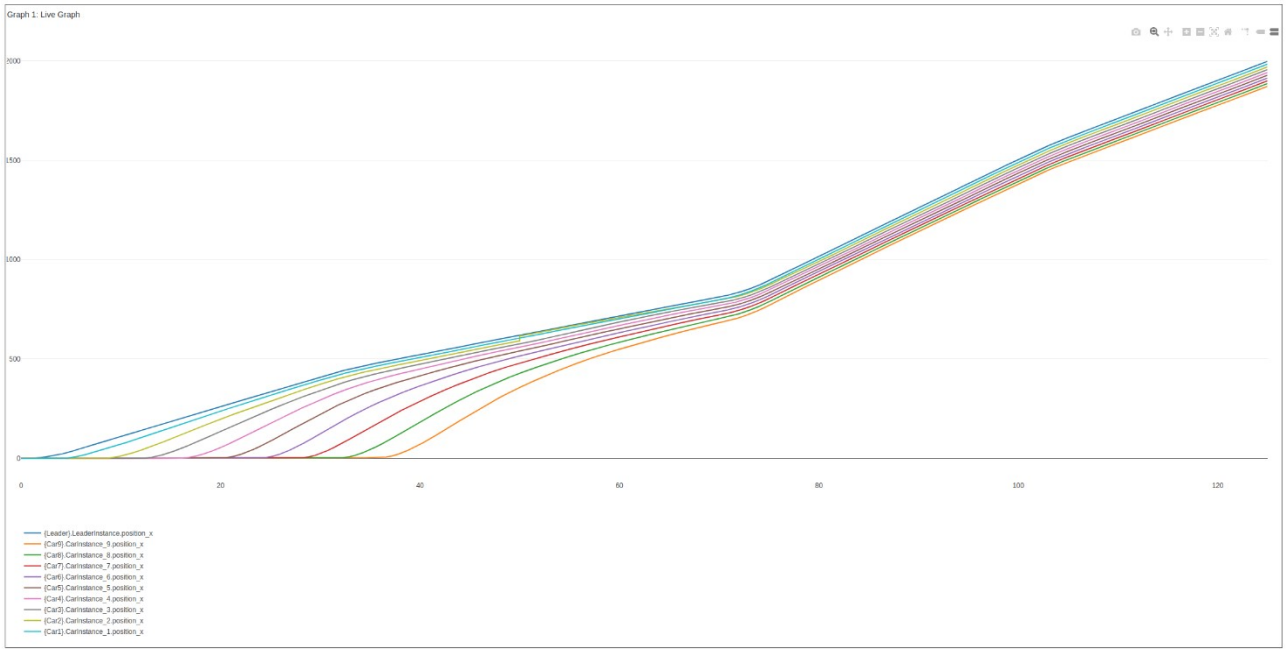

Figure 3: Graph of the position of the vehicles – case 1

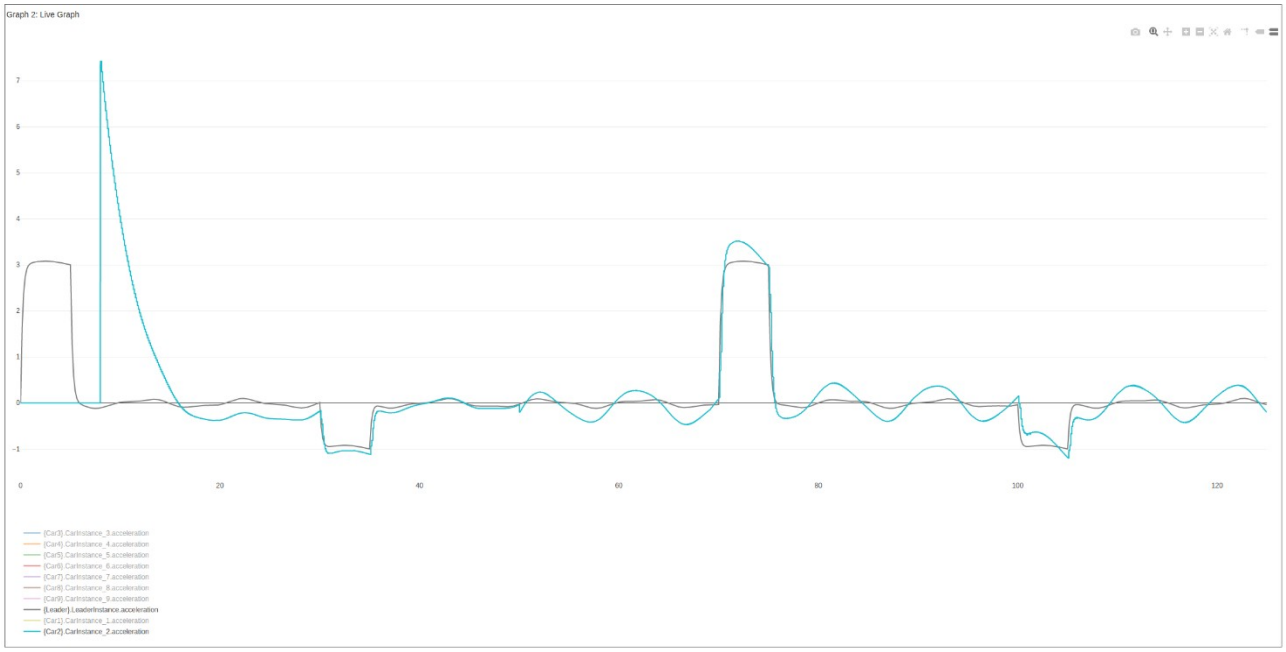

Figure 4: Acceleration of the leader (nominal) and of Car2 (attacked) – case 1

Case 2. We introduce an attack where the desired acceleration sent by the Edge to Car2 is wrong. The type of alteration is a constant increase of the desired acceleration value. Figure 5 shows the effects of an actuator attack. Figure 6 shows the value of the desired acceleration sent by the Edge to Car2 which is modified by adding a constant value.

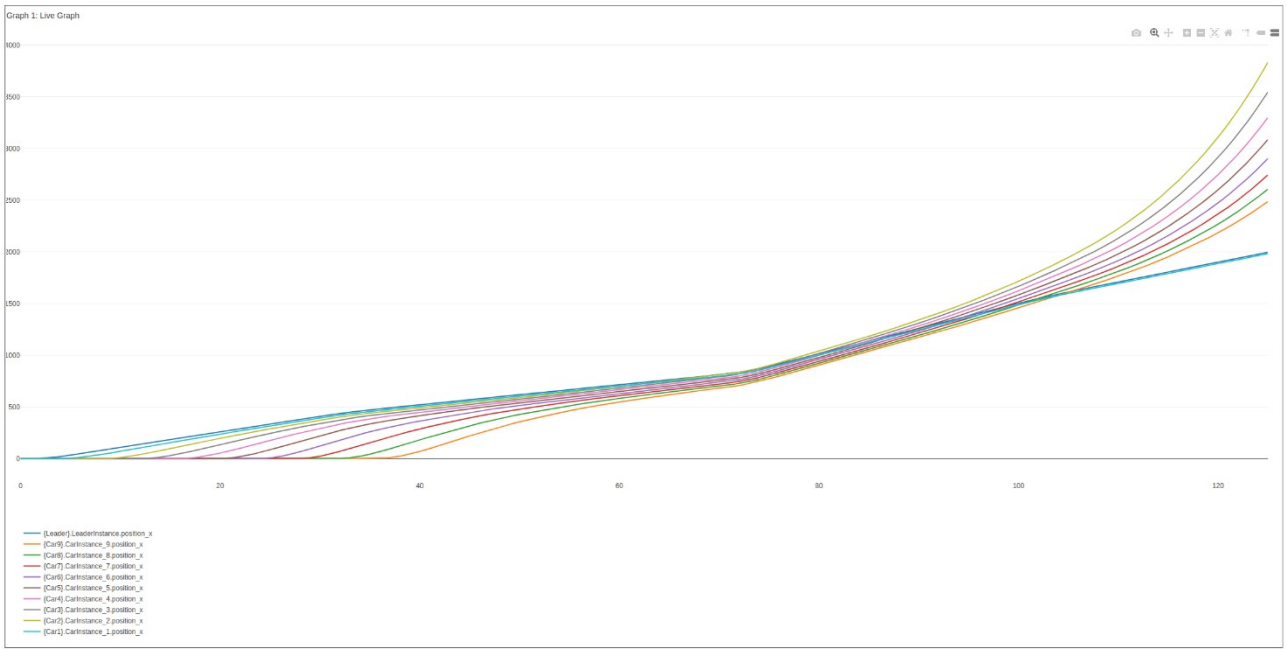

Figure 5: Graph of the position of the vehicles – case 2

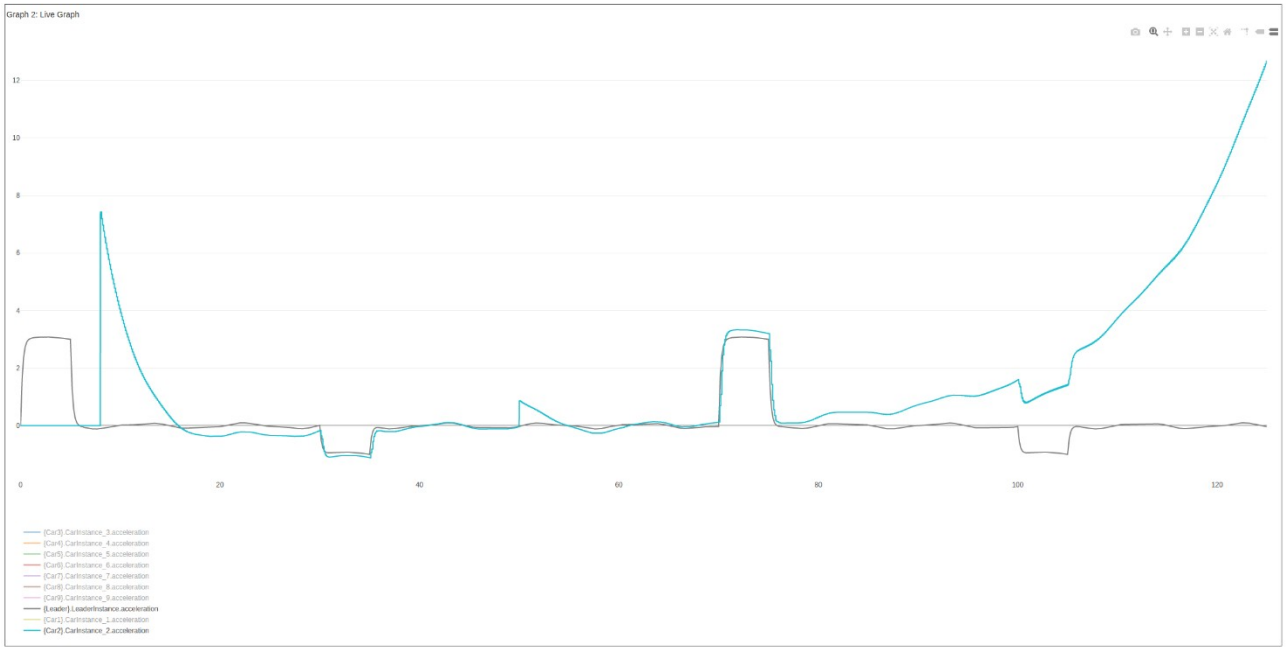

Figure 6: Acceleration of the leader (nominal) and of Car2 (attacked) – case 2

After Task 2.1: “Implementation and testing of CPS model”, an extensive simulation campaign has been conducted by performing a Design Space Exploration to generate behavioural traces for different nominal operational conditions of the platoon, Task T2.2: “Design Space Exploration and trace generation”. Then in Task T2.3 “Implementation and testing of attacks (FMUs)”, the attacks were implemented in the CPS architecture, attacking the critical points identified in Task 1.3. Finally, in Task T2.4 “Design Space Exploration & trace with attacks generation” simulations were conducted for the identified attack scenarios. Collected data are organised in .csv files, and are available on the Foreseen machine of the FoReLab at RU-PI. Part of the data will also be uploaded on the website of the project. The collected data will be used in next steps of the project to build formal models of the CPS behaviour in absence and presence of attacks

7. Roadmap for future work

WP2 is the work package in charge of executing the co-simulation runs and collect data required for the next step of the project. In particular, in the referenced period of the project, the FMI architecture has been implemented to execute a validated campaign of co-simulation runs and a dataset of traces with and without the attacks has been saved. The objective of WP3 will be to formalize platoon behavior by transforming the execution traces into formal models that represent both normal and malicious system dynamics. These models will account for complex interactions, such as cascading effects of attacks and varying network conditions. Embedded detection mechanisms will enhance the system's ability to identify recurring malicious patterns or anomalies. Furthermore, the models will try to incorporate predictive capabilities to anticipate potential threats.

8. Bibliography

[MathWork] MathWorks. Simulink. url: https://it.mathworks.com/help/simulink/.

[Gam2017] Gamble, C., “DSE in the INTO-CPS Platform”. INTO-CPS Deliverable, Aarhus Univ., Denmark, Tech. Rep. D5.3e

[Ber2020] Bernardeschi C., Domenici A., Palmieri M. “Formalization and co-simulation of attacks on cyber-physical systems,” Journal of Computer Virology and Hacking Techniques, 16, 2020