MISSIONE 4
ISTRUZIONE
RICERCA

FORESEEN

Deliverable D5.2: Final report on the application of the FORESEEN to the use case

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 mEthodS for attack dEtEction in autonomous driviNg systems
PRIN 2022 PNRR

Project number: P2022WYAEW
CUP: I53D23006130001

Deliverable D5.2: Final report on the application of the FORESEEN to the use case

Project Start Date: 30/11/2023 Duration: 24 months
Coordinator: *University of Pisa*

Deliverable No	**D5.2**
WP No:	WP3
WP Leader:	RU-MI
Tasks:	T3.7 Validation through co-simulation on use case
Due date:	M 21-24
Delivery date:	February 23, 2026
Authors:	RU-MI, RU-PI, RU-PA, 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

ADAS	Advanced Driver-Assistance Systems
CACC	Cooperative Adaptive Cruise Control
CAN	Controller Area Network
CCS	Calculus of Communicating Systems
CSV	Comma-Separated Values
CWB-NC	Concurrency WorkBench of the New Century
DSRC	Dedicated Short-Range Communications
FDR	False Discovery Rate
FMU	Functional Mock-up Unit
FNR	False Negative Rate
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
PID	Proportional-Integral-Derivative
RGB	Red, Green, Blue
TNR	True Negative Rate
TPR	True Positive Rate
V2E/V2N	Vehicle to Edge/Vehicle to Network (synonyms for this deliverable)
V2V	Vehicle to Vehicle

1. Introduction

This deliverable presents the results of WP3 (Task 3.7), the main objective of which is to validate the methodology developed in the project on the platooning use case. This validation is conducted using the co-simulation framework developed in WP2 and the system properties selected in WP3 (Task 3.2, Task 3.3) and expressed in mu-calculus in Task 3.6 (Deliverable 5.1). The tool chain implemented for generation of traces and verifying properties is described, and the results of applying the tools to the platoon, taking into account traces obtained from the different operational modes of the leader, are presented. The results are aggregated by the high-level scenario being simulated (e.g., "No Attack", "Attack on car 1's sensors"), the value of the attacks, and the property being checked. Each property is a formally specified safety condition on the platoon's inter-vehicle distances and accelerations. The percentage of simulation traces, within the batch, for which the property evaluates to True or False is shown. As every property is expressed as the negation of a safety condition, a True outcome indicates that an attack is occurring, potentially, whereas a False outcome indicates that the safety condition held throughout, i.e., no violations were observed. This analysis allows us to assess the detection performance of each formula. Additionally, an example is provided on how the model checking technique supports the interpretation of anomalous behaviors detected in the analyzed system traces. Finally, an alternative car physics simulator, that also provides a graphical representation of the simulated platoon implemented in the project, is presented.

2. Automated tool for properties verification

To automate the verification of formal properties on the simulation traces, we developed a system that allows the execution of the Concurrency WorkBench [Cleaveland, 2000] over multiple models automatically. The complete source code for the helper tools is available at https://github.com/ForeseenPRIN/cwb_traces_generator .

2.1. Generation of traces from the model

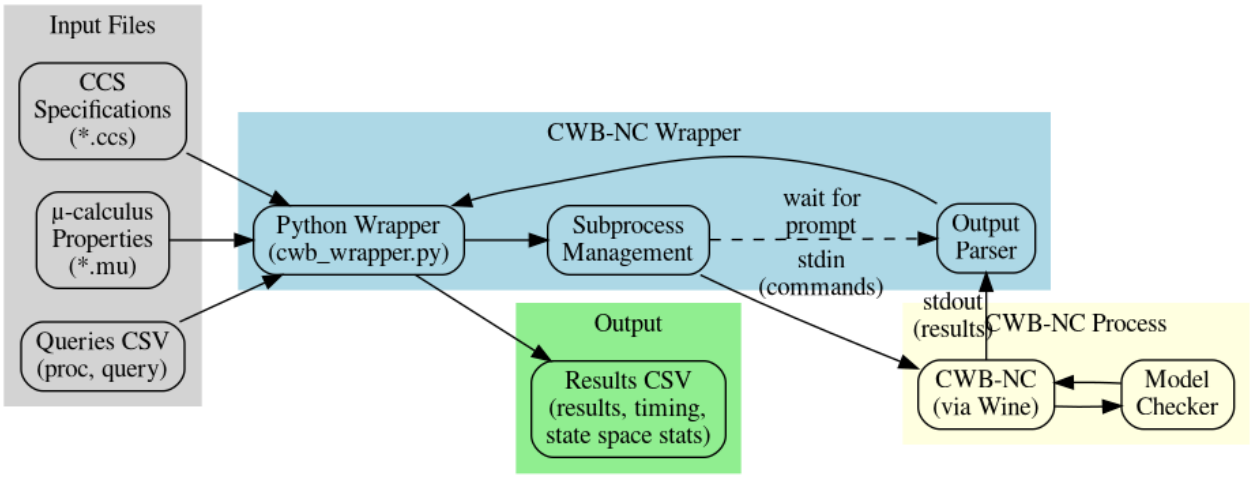
As already reported in the previous deliverables, to generate a CWB automaton from a simulation trace, we process each simulation trace as follows:

1. We sample the trace at a lower frequency to reduce the number of tuples in trace
2. We discretize the real values from the simulation into discrete values
3. We generate a CCS automaton in which those discrete values appear in order

2.2. Running the queries

The CWB-NC Wrapper is a Python-based automation tool developed to streamline the verification process of concurrent systems modeled in CCS (Calculus of Communicating Systems) [Mil89] against temporal properties expressed in μ-calculus [Koz83]. This tool addresses the limitations of manual interaction with the Concurrency WorkBench of the New Century (CWB-NC) by providing a programmatic interface that enables batch processing, automated result collection, and comprehensive statistical analysis.

The wrapper implements a process-based communication architecture that bridges Python with the CWB-NC tool running under Wine. The system architecture is illustrated in Figure below.



We give as input to the wrapper the ccs and mu files containing the definitions of the automata and the properties, respectively; and a CSV file containing the queries to run.

The queries CSV file serves as the batch specification for verification tasks. It uses a simple two-column format that pairs processes with the properties to be verified. This format enables systematic verification of multiple process-property combinations without manual intervention.

The CSV must have two columns:

- The “proc” column contains the names of the CCS process agent to be verified. This must exactly match a process identifier defined in the loaded CCS specification files (`.ccs` files). Process names are case-sensitive and must follow CWB-NC's naming conventions.
- The “query” column contains the names of the μ-calculus properties to check against the specified process. This identifier must correspond to a property defined in the loaded μ-calculus specification files (`.mu` files). The property name is also case-sensitive.

The batch of queries described in the file are then processed by running, for each of them, the CWB-NC command

```
chk process property
```

The results are then collected into an output CSV that contains the result of each query. It contains the following columns:

- run_name: Experiment identifier for grouping results
- proc: Process that was verified
- query: Property that was checked
- result: Verification outcome (TRUE/FALSE)
- states: Number of states in the model's state space
- transitions: Number of transitions explored
- user, system, gc, real: Time measurements in seconds, as provided by CWB-NC

The implementation of this utility can be found at the following link: https://github.com/ForeseenPRIN/cwb_traces_generator .

2.3. Implementation

We wrote a GNU Make Makefile that glues everything together and allows the various jobs to be executed concurrently and thus exploiting the parallelism of modern multi-core microprocessors.

The final results are stored in a CSV file that contains, for each run, the result of each query.

The used Makefile is available in Appendix A.

3. Results

Before showing the results, let's recapitulate the properties we want to evaluate on the traces.

To save space in this summary, *substitutions* for similar formulae are used: a substitution like a{b,c} is a compact notation where the part inside the curly braces lists several alternatives, each of which gets independently combined with the surrounding fixed parts. So, a{b,c} expands to ab and ac. When two such groups appear together, like {G,R}C{1,2}, every combination of choices is produced: GC1, GC2, RC1, RC2.

The group {G,R} differentiates between *GPS* and *radar* readings, respectively, for the value of the inter-vehicular distance between two cars. The group {1,2} is used to specify in the formula that applies to car 1 or car 2; for the former, the inter-vehicular distance between car 1 and the leader is used, for the latter, the inter-vehicular distance between car 2 and car 1 is used.

- `not is_d_safe_{G,R}C{1,2}` Detects whether car1 or car2's GPS or RADAR distance reading remains in a low-or-critical range for a sustained number of consecutive steps, signaling a persistent unsafe-distance condition for either sensor on either car.
- `not is_d_safe_w_dv_{G,R}C1` Extends the GPS/RADAR unsafe-distance check for car1 by also requiring that the speed differential remains opening (gap widening) across some steps, thus combining distance and relative-velocity evidence of an anomaly.
- `not is_d_qsafe_{G,R}C{1,2}` Like is_d_safe but restricted to only the *critically* low distance level (stricter threshold), detecting a sustained critically-close reading on either sensor for either car.
- `not is_d_qsafe_w_dv_{G,R}C1` The strictest combined check for car1's RADAR: detects a sustained *critically* low distance together with an opening speed differential, requiring both conditions to persist over consecutive steps.
- `not is_acc_safe_w_d_unsafe_RC1` Detects whether car1's RADAR reports a persistently unsafe distance while car1's acceleration remains suspiciously unresponsive (no braking) for an extended sequence of steps.
- `not is_acc_safe_w_d_unsafe_RC1_v2` A variant of the above but with different timing
- `not is_acc_safe_w_d_crit_RC1` Detects whether car1's RADAR reports a persistently *critically* unsafe distance while car1's acceleration remains suspiciously unresponsive (no braking) for an extended sequence of steps.

A more complete description of the formulae is available in Deliverable 5.1.

On the GPS-computed vs RADAR distances, the former is the one that is used by the CACC control law to govern the platoon; whereas the latter is used only internally by the vehicle and it's left unattacked. As described in the previous deliverables.

3.1. Vehicle to Edge

Let us consider only a platoon of 3 cars (leader + 2 followers) on all the generated traces. Let us consider the case of V2E communication

Our generation tool builds the distance, speed difference and acceleration labels from the message containing the readings of the sensors and sent to the edge. These messages have been altered by the attack; thus the ok query is always true.

The results are presented, aggregated by batch and attack, in the tables in the following subsections, whose columns' meanings are the following:

The **batch_name** column identifies the experiment group, i.e., the high-level scenario being simulated (for example, "No Attack", "Attack on car 1's sensors"). All rows sharing the same batch_name belong to the same set of simulation runs and differ only in the value of attack and the property being checked.

The **attack** column is a parameter of the FMU components that selects which type of attack, if any, is injected into the simulation. Its meaning differs slightly depending on which vehicle is being attacked (leader vs. follower) and on what part of the vehicle is targeted (sensors vs. actuator). In the next subsections the specific meaning of the values of attack will be explained; a more detailed explanation is available in the previous deliverables.

The **query** column names the μ-calculus property that was verified by the CWB-NC tool against the CCS automaton generated from the simulation trace. Each property is a formally specified safety condition on the platoon's inter-vehicle distances and accelerations; a full description of each formula is provided in Deliverable D5.1.

The **False** and **True** columns report the percentage of simulation traces, within the batch, for which the property evaluated to False or True respectively. Because every property is expressed as the *negation* of a safety condition (e.g., not is_d_safe_GC1), a **True** outcome means that an attack is happening, potentially, while a **False** outcome means the safety condition held throughout, i.e., no violation was observed. The two percentages always sum to 100%.

No Attacks

batch_name	attack	query	False	True
No Attack. The nominal case	0	not is_acc_safe_w_d_unsafe_RC1	65.63%	34.38%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	100.00%	0.00%
		not is_d_safe_GC1	60.42%	39.58%
		not is_d_safe_GC2	60.42%	39.58%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	98.96%	1.04%
		not is_d_qsafe_GC2	100.00%	0.00%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_safe_RC1	60.42%	39.58%
		not is_d_safe_RC2	60.42%	39.58%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	98.96%	1.04%
		not is_d_qsafe_RC2	100.00%	0.00%
		not is_d_qsafe_w_dv_RC1	100.00%	0.00%

In this table the attack column is always **0**, meaning no attack is active. The simulation runs under purely nominal conditions: no spurious signal is added to any sensor output or actuator input in any vehicle. This serves as the baseline against which all other scenarios are evaluated.

- **0** — No attack. All sensor readings (position, speed, acceleration) reflect the true physical state of the vehicle without any modification.

The results presented in the table provide an overview of the system's behaviour under nominal (attack free) operating conditions. As expected in a baseline scenario, most safety related properties are consistently satisfied, indicating stable platoon behaviour and correct operation of the Cooperative Adaptive Cruise Control (CACC) mechanisms.

A first observation concerns the **acceleration related properties**. Both *not is_acc_safe_w_d_unsafe_RC1_v2* and *not is_acc_safe_w_d_crit_RC1* report 100% "False" outcomes, meaning that their safety conditions are always satisfied in the nominal case. This confirms that the vehicle's acceleration remains well within safe limits whenever the distance is either unsafe or critically unsafe, aligning with expected behaviour for a correctly functioning controller. The property *not is_acc_safe_w_d_unsafe_RC1*, however, shows a 34.38% "True" count, indicating that in roughly one third of traces the acceleration condition becomes relevant enough to trigger the property. This does not necessarily reflect unsafe behaviour; rather, it highlights instances where the system transitions near threshold values, which is normal given sensor noise and vehicle dynamics.

Regarding **distance based properties**, both *not is_d_qsafe_RC1* and *not is_d_qsafe_RC2* display a balanced split (≈60% False, 40% True). This pattern is mirrored by their GPS based counterparts (*GC1* and *GC2*). These results indicate occasional proximity of vehicles to the predefined "qsafe" distance thresholds, but without persistent violations. Oscillations around safety margins are expected in realistic CACC scenarios, especially considering variability in speed, intervehicle communication latency, and model discretisation.

A notable difference emerges in the behaviour of the **quasi--safe variants**, such as *not is_d_qsafe_GC1* (98.96% False), *not is_d_qsafe_RC1* (98.96% False), and the corresponding car2

entries (100% False). These properties use **stricter thresholds** and therefore very rarely signal a violation when no attack is present. This confirms their suitability as high precision detectors, producing virtually no- false alarms in benign conditions.

Similarly, all **speed-difference-related properties** (e.g., *not is_d_safe_w_dv_GC1*) consistently return 100% "False". Since no attack affects the speed differential in the baseline scenario, the system behaves deterministically, and the distance–speed relationship remains within expected ranges. These results support the conclusion that these properties are too conservative to detect attacks in more complex scenarios but are perfectly stable under nominal conditions.

Overall, the table validates that the system behaves as intended without attacks. Properties with higher "True" rates correspond to those more sensitive to natural variations in distance and acceleration, while stricter properties demonstrate excellent stability and no false alarms. This baseline will serve as a reference for evaluating deviations observed in subsequent attack scenarios.

Attack on car 1's sensors

batch_name	attack	query	False	True
Attack on car 1's sensors	1	not is_acc_safe_w_d_unsafe_RC1	66.67%	33.33%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	100.00%	0.00%
		not is_d_qsafe_GC1	0.00%	100.00%
		not is_d_qsafe_GC2	59.38%	40.63%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	8.33%	91.67%
		not is_d_qsafe_GC2	100.00%	0.00%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	62.50%	37.50%
		not is_d_qsafe_RC2	0.00%	100.00%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	100.00%	0.00%
		not is_d_qsafe_RC2	0.00%	100.00%
		not is_d_safe_w_dv_rC1	100.00%	0.00%
	2	not is_acc_safe_w_d_unsafe_RC1	46.88%	53.13%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	100.00%	0.00%
		not is_d_qsafe_GC1	57.29%	42.71%
		not is_d_qsafe_GC2	58.33%	41.67%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	100.00%	0.00%
		not is_d_qsafe_GC2	100.00%	0.00%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	41.67%	58.33%
		not is_d_qsafe_RC2	66.67%	33.33%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	100.00%	0.00%
		not is_d_qsafe_RC2	100.00%	0.00%

batch_name	attack	query	False	True
		not is_d_safe_w_dv_rC1	100.00%	0.00%
	3	not is_acc_safe_w_d_unsafe_RC1	68.75%	31.25%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	100.00%	0.00%
		not is_d_qsafe_GC1	0.00%	100.00%
		not is_d_qsafe_GC2	60.42%	39.58%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	8.33%	91.67%
		not is_d_qsafe_GC2	100.00%	0.00%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	62.50%	37.50%
		not is_d_qsafe_RC2	0.00%	100.00%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	100.00%	0.00%
		not is_d_qsafe_RC2	0.00%	100.00%
		not is_d_safe_w_dv_rC1	100.00%	0.00%

Here, the attack parameter is applied to the **sensors** of follower car 1 (implemented in the SimpleCar FMU). When active, the attack function injects a spurious acceleration signal on top of the vehicle's sensor outputs — position, speed and acceleration are all altered coherently. The attack begins at the time specified by attack_time and persists until the end of the simulation. Two auxiliary sinusoidal functions drive the corruption:

- $\texttt{low\_freq\_sine(t)} = \texttt{attack\_amplitude} \cdot \sin(\texttt{low\_frequency} \cdot 2\pi \cdot \texttt{t})$
- $\texttt{high\_freq\_sine(t)} = \texttt{attack\_amplitude} \cdot \sin(\texttt{high\_frequency} \cdot 2\pi \cdot \texttt{t})$

The three active values correspond to:

- **1** — Low-frequency sensor attack. low_freq_sine(t) is added to the sensor outputs of car 1, producing a slowly oscillating bias in the perceived distance and speed that is broadcast to the CACC network.
- **2** — High-frequency sensor attack. high_freq_sine(t) is added instead, injecting a rapidly oscillating noise component into the sensor readings.
- **3** — Combined sensor attack. Both low_freq_sine(t) and high_freq_sine(t) are summed and added to the sensor outputs, producing a composite waveform that combines slow drift and rapid oscillation.

The results for the attack on car 1's sensors reveal a clear and consistent detection pattern across all three attack intensity levels. The most striking finding is the behaviour of the distance-based properties targeting car 1: *not is_d_qsafe_GC1* and *not is_d_qsafe_RC2* both achieve 100% True rates in attacks 1 and 3, indicating that the GPS and RADAR distance readings of car 1 are severely and persistently distorted by the attack, producing readings that fall well within the critically unsafe range for every trace. Attack 2, which appears to be a weaker or differently timed variant, results in a more mixed split (approximately 40–58% True), suggesting the attack's effect on the distance signal is less systematic.

Notably, all speed-differential properties (*not is_d_safe_w_dv_**) return 100% False across all three attack levels, confirming that — as already observed in the nominal case — these properties are un-

able to detect this class of attack, since the relative speed between vehicles is not sufficiently affected. Similarly, both *not is_acc_safe_w_d_unsafe_RC1_v2* and *not is_acc_safe_w_d_crit_RC1* consistently return 100% False, indicating that the acceleration-plus-critical-distance combined check is not triggered: the CACC controller continues to respond to the falsified sensor readings in a way that keeps the acceleration within expected bounds, masking the attack from these detectors.

The acceleration property *not is_acc_safe_w_d_unsafe_RC1* shows moderate True rates (31–53%), indicating partial detectability of the attack from the acceleration signal alone, but with considerable variability. Overall, the most reliable indicators for this attack type are the car1-targeted distance properties (*not is_d_qsafe_GC1* and *not is_d_qsafe_RC2*), while car2-targeted and speed-differential properties offer little to no detection capability.

Attack on Car 1's actuators

batch_name	attack	query	False	True
Attack on car 1's actuators	4	not is_acc_safe_w_d_unsafe_RC1	48.96%	51.04%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	95.83%	4.17%
		not is_d_qsafe_GC1	47.92%	52.08%
		not is_d_qsafe_GC2	56.25%	43.75%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	93.75%	6.25%
		not is_d_qsafe_GC2	100.00%	0.00%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	47.92%	52.08%
		not is_d_qsafe_RC2	56.25%	43.75%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	93.75%	6.25%
		not is_d_qsafe_RC2	100.00%	0.00%
		not is_d_safe_w_dv_rC1	100.00%	0.00%
	5	not is_acc_safe_w_d_unsafe_RC1	0.00%	100.00%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	3.13%	96.88%
		not is_d_qsafe_GC1	0.00%	100.00%
		not is_d_qsafe_GC2	62.50%	37.50%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	1.04%	98.96%
		not is_d_qsafe_GC2	100.00%	0.00%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	0.00%	100.00%
		not is_d_qsafe_RC2	62.50%	37.50%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	1.04%	98.96%
		not is_d_qsafe_RC2	100.00%	0.00%
		not is_d_safe_w_dv_rC1	100.00%	0.00%

Here, the attack parameter targets the **actuator** of follower car 1, specifically its desired acceleration input. Unlike the sensor attacks above, this attack does not falsify what the car measures or broadcasts.

- **4** — Multiplicative actuator attack. The desired acceleration is scaled by a factor: `u_(t) = u(t) · (1 + attack_amplitude)`, where u is the *desired acceleration* sent by the CACC controller. This amplifies or damps the control command proportionally, making the vehicle over- or under-react to every manoeuvre.
- **5** — Additive actuator attack. A constant bias is added: `u_(t) = u(t) + attack_amplitude`. This shifts the acceleration uniformly, for instance preventing the vehicle from fully braking or causing a persistent unwanted thrust, regardless of what the controller commands.

The attack on car 1's actuators produces a noticeably different detection profile compared to the sensor attack, reflecting the different nature of the fault: rather than corrupting the perceived environment, this attack directly affects the vehicle's physical response.

For the weaker attack variant (attack 4), detection rates are moderate across most properties, with *not is_d_qsafe_GC1* and *not is_d_qsafe_RC1* achieving approximately 52% True, and the acceleration property *not is_acc_safe_w_d_unsafe_RC1* also crossing the 50% threshold. The critical-distance acceleration check *not is_acc_safe_w_d_crit_RC1* shows a small but non-zero True rate (4.17%), which is a significant departure from both the nominal case and the sensor attack scenario, where it was always 100% False. This suggests that a sufficiently aggressive actuator attack does occasionally push car 1 into a critically unsafe following distance while the acceleration remains insufficiently corrective — precisely the scenario this property is designed to detect.

For the stronger attack variant (attack 5), detection rates increase dramatically. Both *not is_d_qsafe_GC1* and *not is_d_qsafe_RC1* reach 100% True, while *not is_d_qsafe_GC1* (strict variant) and *not is_d_qsafe_RC1* (strict variant) approach 99% True. Most strikingly, *not is_acc_safe_w_d_unsafe_RC1* achieves 100% True and *not is_acc_safe_w_d_crit_RC1* reaches

96.88% True — the highest values observed for the acceleration-based properties across all scenarios. This confirms that a strong actuator attack, which suppresses or reverses braking, is most effectively revealed by the combination of a persistently unsafe distance and an unresponsive acceleration signal.

As with all other scenarios, speed-differential properties and *not is_acc_safe_w_d_unsafe_RC1_v2* remain at 100% False, reinforcing their consistent inability to detect any of the tested attack types.

Attack on Leader's sensors

batch_name	attack	query	False	True
	1	not is_acc_safe_w_d_unsafe_RC1	0.00%	100.00%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	0.00%	100.00%
		not is_d_qsafe_GC1	43.75%	56.25%
		not is_d_qsafe_GC2	54.69%	45.31%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	95.31%	4.69%
		not is_d_qsafe_GC2	99.48%	0.52%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	0.00%	100.00%
		not is_d_qsafe_RC2	54.69%	45.31%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	0.00%	100.00%
		not is_d_qsafe_RC2	99.48%	0.52%
		not is_d_safe_w_dv_rC1	100.00%	0.00%
Attack on leader's sensors	2	not is_acc_safe_w_d_unsafe_RC1	40.63%	59.38%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%
		not is_acc_safe_w_d_crit_RC1	77.60%	22.40%
		not is_d_qsafe_GC1	23.44%	76.56%
		not is_d_qsafe_GC2	31.25%	68.75%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	65.63%	34.38%
		not is_d_qsafe_GC2	84.90%	15.10%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	35.94%	64.06%
		not is_d_qsafe_RC2	31.25%	68.75%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	73.44%	26.56%
		not is_d_qsafe_RC2	84.90%	15.10%
		not is_d_safe_w_dv_rC1	100.00%	0.00%
	3	not is_acc_safe_w_d_unsafe_RC1	0.00%	100.00%
		not is_acc_safe_w_d_unsafe_RC1_v2	100.00%	0.00%

batch_name	attack	query	False	True
		not is_acc_safe_w_d_crit_RC1	0.00%	100.00%
		not is_d_qsafe_GC1	26.04%	73.96%
		not is_d_qsafe_GC2	31.25%	68.75%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_GC1	69.27%	30.73%
		not is_d_qsafe_GC2	89.06%	10.94%
		not is_d_safe_w_dv_GC1	100.00%	0.00%
		not is_d_qsafe_RC1	0.00%	100.00%
		not is_d_qsafe_RC2	31.25%	68.75%
		not is_d_safe_w_dv_RC1	100.00%	0.00%
		not is_d_qsafe_RC1	0.00%	100.00%
		not is_d_qsafe_RC2	89.06%	10.94%
		not is_d_safe_w_dv_rC1	100.00%	0.00%

Here, the same sensor attack modes (1, 2, 3) as described for car 1 are applied instead to the **leader vehicle**. Note that the leader's attack function supports only modes 0–3 (no actuator attacks). Because the leader's sensor readings are used by all following vehicles to compute their desired accelerations, an attack at this level has a platoon-wide cascading effect.

- **1** — Low-frequency sensor attack on the leader. `low_freq_sine(t)` is added to the leader's position, speed and acceleration outputs, causing all followers to react to a slowly drifting false reading of the lead vehicle's state.
- **2** — High-frequency sensor attack on the leader. `high_freq_sine(t)` is added, injecting high-frequency noise into the leader's sensor outputs.
- **3** — Combined sensor attack on the leader. Both sinusoidal components are injected simultaneously into the leader's sensors.

The attack on the leader's sensors presents the most distinctive and, in several respects, the most severe detection profile among all tested scenarios. Because the leader's sensor readings directly influence the speed and acceleration commands broadcast to all following vehicles, an attack at this level has a cascading effect on the entire platoon, which is reflected in the broader spread of positive detections across both car1 and car2 targeted properties.

For attacks 1 and 3 — the stronger intensity levels — both *not is_d_qsafe_RC1* and *not is_d_qsafe_RC1* (strict variant) achieve 100% True, while the acceleration property *not is_acc_safe_w_d_unsafe_RC1* and *not is_acc_safe_w_d_crit_RC1* also reach 100% True. This is the only scenario in which the critical-distance acceleration check fires at full rate, highlighting that the leader-level attack is uniquely capable of simultaneously driving car 1 into a critically close following distance and causing its acceleration to become dangerously unresponsive. The GPS-based properties for car 1 show high True rates as well (around 56–74%), though they are somewhat lower than their RADAR counterparts, suggesting a degree of sensor-type asymmetry in how the attack manifests.

For attack 2, a weaker or differently parameterised variant, detection rates drop considerably across all properties (22–64% True), indicating that a less aggressive leader-sensor manipulation is harder

to distinguish from normal transient behaviour. Car 2 properties consistently show lower True rates than car 1 properties across all three attacks, which is expected given the additional buffering provided by car 1's own dynamics.

Once again, all speed-differential and *not is_acc_safe_w_d_unsafe_RC1_v2* properties return 100% False throughout, and the GPS quasi-safe car 2 properties (*not is_d_qsafe_GC2*) are very close to zero True (0.52% and 10.94%), confirming their near-total insensitivity to this attack type.

Statistical indicators

False Discovery Rate (FDR) is the fraction of detections reported as "positive" by a property that are actually incorrect (i.e., false positives).

$$\text{FDR} = \frac{\text{False Positives}}{\text{True Positives} + \text{False Positives}}$$

A low FDR means that when the property signals an attack, it is usually correct. A high FDR indicates many false alarms. This value is related to the ***precision*** as it is its opposite, that is the precision is $P = 1 - FDR$.

False Negative Rate (FNR) is the fraction of attacks that occurred but were *not* detected by the property.

$$\text{FNR} = \frac{\text{False Negatives}}{\text{True Positives} + \text{False Negatives}}$$

A low FNR means the detector rarely misses attacks. A high FNR indicates that the property often fails to notice an ongoing attack.

True Positive Rate (TPR) (*also called* ***recall*** *or* ***sensitivity)*** is the proportion of actual attacks that are correctly detected by the property.

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

A high TPR indicates strong detection capability: most attacks trigger the property as expected.

True Negative Rate (TNR) *(also called **specificity**)* is the proportion of benign (noattack) scenarios correctly identified as safe.

$$\text{TNR} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

A high TNR means the property rarely triggers during normal attack-free operation.

Let us now consider the rate of false and true positive and negatives ratios. The figures in the following table were computed by considering simulation traces acquired so far, as explained in the previous sections of this document. Only the previously selected properties are considered.

query	*False discovery rate (1-precision)*	*False negative rate*	*True positive rate (recall)*	*True negative rate (specificity)*
not is_d_qsafe_RC1	4.64%	26.04%	73.96%	60.42%
not is_d_qsafe_GC1	4.67%	26.52%	73.48%	60.42%
not is_acc_safe_w_d_unsafe_RC1	4.18%	28.41%	71.59%	65.62%
not is_d_qsafe_RC2	5.50%	38.16%	61.84%	60.42%
not is_d_qsafe_GC2	6.51%	48.30%	51.70%	60.42%
not is_d_qsafe_RC1	0.19%	49.24%	50.76%	98.96%
not is_acc_safe_w_d_crit_RC1	0.00%	50.38%	49.62%	100.00%
not is_d_qsafe_GC1	0.24%	61.08%	38.92%	98.96%
not is_d_qsafe_RC2	0.00%	76.99%	23.01%	100.00%
not is_d_qsafe_GC2	0.00%	95.17%	4.83%	100.00%
not is_acc_safe_w_d_unsafe_RC1_v2	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_GC1	0.00%	100.00%	0.00%	100.00%

not is_d_safe_w_dv_GC1	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_RC1	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_rC1	0.00%	100.00%	0.00%	100.00%

The property `not is_acc_safe_w_d_unsafe_RC1` demonstrated strong overall detection performance with 756 true positives and only 33 false positives, yielding a false discovery rate of 4.18%. However, it recorded 300 false negatives, resulting in a false negative rate of 28.41% and a true positive rate of 71.59%. This indicates that while the property reliably detects attacks when triggered, it misses approximately 28% of actual attack cases.

Among distance-based properties, `not is_d_qsafe_RC1` achieved the highest true positive rate at 73.96% with 781 true positives, while maintaining a low false discovery rate of 4.64% (38 false positives out of 819 total detections). The GPS equivalent `not is_d_qsafe_GC1` performed similarly with 776 true positives and a 73.48% detection rate, though with a slightly higher false discovery rate of 4.67%. Both properties demonstrated comparable false negative rates around 26%, indicating they miss roughly one quarter of attack instances.

The car2 variants of distance properties showed notably different performance characteristics. The property `not is_d_qsafe_RC2` achieved 653 true positives with a 61.84% detection rate and a false negative rate of 38.16%, while `not is_d_qsafe_GC2` recorded only 546 true positives with a 51.70% detection rate and 48.30% false negative rate. Both car2 properties maintained low false discovery rates (5.50% and 6.51% respectively) but exhibited significantly higher miss rates compared to their car1 counterparts.

The quasi variants, which enforce stricter thresholds, exhibited substantially different detection characteristics. The property `not is_d_qsafe_RC1` recorded 536 true positives with a 50.76% true positive rate and 49.24% false negative rate, achieving an exceptionally low false discovery rate of 0.19% with only 1 false positive across all traces. The GPS equivalent `not is_d_qsafe_GC1` showed 411 true positives with a 38.92% detection rate but a higher 61.08% false negative rate, while maintaining a similarly excellent false discovery rate of 0.24%.

The car2 quasi variants demonstrated markedly poor detection performance. The property `not is_d_qsafe_RC2` detected only 243 attacks (23.01% true positive rate) with 813 false negatives

(76.99% false negative rate), while `not is_d_qsafe_GC2` achieved merely 51 true positives (4.83% detection rate) with 1005 false negatives (95.17% false negative rate). However, both maintained perfect specificity with 100% true negative rates and zero false positives.

The acceleration quasi property `not is_acc_safe_w_d_crit_RC1` recorded 524 true positives but 532 false negatives, yielding a 49.62% detection rate and 50.38% false negative rate. With zero false positives, it achieved a 0% false discovery rate and perfect 100% specificity, indicating it never triggers on benign scenarios but misses approximately half of actual attacks.

Three categories of properties demonstrated complete inability to detect attacks: all speed differential properties (`*_w_diff_speed_*`), the `not is_acc_safe_w_d_unsafe_RC1_v2` property, and the GPS quasi speed differential variant. These properties recorded zero true positives and 1056 false negatives each, resulting in 100% false negative rates and 0% true positive rates. Conversely, they achieved perfect specificity with 96 true negatives and zero false positives, yielding 100% true negative rates and 0% false discovery rates. This indicates these properties are overly conservative, never triggering regardless of attack presence, and therefore provide no practical detection capability despite maintaining perfect precision when they do trigger (which is never).

All properties demonstrated strong specificity in identifying true negatives, with true negative rates ranging from 60.42% to 100%. The quasi variants achieved near-perfect or perfect specificity: both `not is_d_qsafe_RC1` and `not is_d_qsafe_GC1` recorded 98.96% true negative rates with 95 true negatives and only 1 false positive each, while all car2 quasi variants and non-detecting properties achieved 100% true negative rates. The standard distance and acceleration properties maintained respectable 60-66% true negative rates, indicating they correctly identify the absence of attacks in roughly two-thirds of benign scenarios, with the remaining third producing false alarms.

Selected properties

Based on the classification metrics, the following properties demonstrate the most effective attack detection capabilities, ranked by their balance of high detection rates and low false alarm rates:

1. `not is_d_qsafe_RC1` - The best overall performer with 781 true positives, achieving a 73.96% detection rate while maintaining only 4.64% false discovery rate. This property successfully detects nearly three quarters of all attacks with minimal false alarms (38 false positives across all traces). Its false negative rate of 26.04% indicates it misses approximately one quarter of attack instances.
2. `not is_d_qsafe_GC1` - Nearly equivalent performance to its radar counterpart with 776 true positives and 73.48% detection rate. The false discovery rate of 4.67% remains exceptionally low, demonstrating that GPS-derived distance measurements provide equally reliable attack detection when monitoring the first follower vehicle. False negative rate of 26.52% is marginally higher than the radar variant.
3. `not is_acc_safe_w_d_unsafe_RC1` - Strong detection capability with 756 true positives and 71.59% true positive rate. This acceleration-focused property achieves the lowest false discovery rate at 4.18% (only 33 false positives), making it the most precise detector in this tier. The 28.41% false negative rate indicates it misses slightly more attacks than the distance-based car1 properties but maintains superior precision.
4. `not is_d_qsafe_RC2` - Solid performance monitoring the second follower vehicle with 653 true positives and 61.84% detection rate. False discovery rate of 5.50% remains acceptable, though the 38.16% false negative rate indicates this property misses more than a third of attack instances affecting car2.
5. `not is_d_qsafe_GC2` - Moderate detection capability with 546 true positives and 51.70% true positive rate. While maintaining a reasonable 6.51% false discovery rate, the 48.30% false negative rate reveals that this property misses nearly half of all attacks on the second follower vehicle, making it the weakest performer in this tier.

3.2. Vehicle to Vehicle

The results for the *vehicle to vehicle* decentralized control scheme are quite similar, thus – for brevity's sake – we report only the final results in tabular form:

query	*False discovery rate (1-precision)*	*False negative rate*	*True positive rate (recall)*	*True negative rate (specificity)*
not is_d_qsafe_RC1	4.64%	26.04%	73.96%	60.42%
not is_d_qsafe_GC1	4.67%	26.52%	73.48%	60.42%
not is_acc_safe_w_d_unsafe_RC1	4.18%	28.41%	71.59%	65.62%
not is_d_qsafe_RC2	5.50%	38.16%	61.84%	60.42%
not is_d_qsafe_GC2	6.51%	48.30%	51.70%	60.42%
not is_d_qsafe_RC1	0.19%	49.24%	50.76%	98.96%
not is_acc_safe_w_d_crit_RC1	0.00%	50.38%	49.62%	100.00%
not is_d_qsafe_GC1	0.24%	61.08%	38.92%	98.96%
not is_d_qsafe_RC2	0.00%	76.99%	23.01%	100.00%
not is_d_qsafe_GC2	0.00%	95.17%	4.83%	100.00%
not is_acc_safe_w_d_unsafe_RC1_v2	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_GC1	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_GC1	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_RC1	0.00%	100.00%	0.00%	100.00%
not is_d_safe_w_dv_rC1	0.00%	100.00%	0.00%	100.00%

3.3. From selected properties to monitoring

The statistical analysis presented in the previous sections identified a subset of formal properties that consistently achieve high detection rates while keeping false alarm rates low, both in the Vehicle-to-Edge and Vehicle-to-Vehicle communication schemes. Building on those findings, we now describe how the best-performing properties can be translated into an online intrusion-detection module, either at the edge server or on board each vehicle, and raises an alert whenever a property violation is detected. Of course, certain properties, such as the one based on the radar readings, may be present only on the vehicle; whereas properties that use things like the GPS-computed distance, may be placed on both the vehicle and/or the edge.

By considering the results coming from the study of the simulation traces through the formal methods and the New Century's Concurrency Workbench, we selected the following properties for the online testing of attacks:

The **distance never too close**, from the properties is d_safe_RC1 and d_safe_GC1, can be formulized, to check for attack at the current time t as

$$T(t) = \exists \bar{t} > 2\text{s} : \forall \tau \in [t - \bar{t}, t]\, d(\tau) \leq LOW.$$

Where $d(t)$ is either the distance read from a radar or by computing the distance between the GPS coordinates of a pair of cars. Ideally, the test T becomes true after two seconds of one of the cars being attacked.

The **don't gain speed (acceleration) when too close**, from the properties such as is_acc_safe_w_d_unsafe_RC1, can be formulized, to check for attack at the current time as

$$T(t) = \forall \tau \in [t - 1.5\text{s}, t]\, d(\tau) < OPT \Rightarrow$$
$$\exists \bar{t} > 1.5\text{s} : \forall \tau \in [t - \bar{t} - 1.5\text{s}, t - 1.5\text{s}]\, d(\tau) < OPT \wedge a(\tau) \geq HIGH.$$

Where $d(\tau)$ is the distance read from a radar.

3.4. Explainability

To support the interpretation of anomalous behaviors detected in the analyzed system traces, we rely on the verification of safety properties over the execution traces produced by the platooning system. Property verification provides an interpretable link between the observed system behavior and the underlying safety requirements, thus helping to explain why a given trace is classified as anomalous or attack-related. In this context, property violations can be used not only to detect unsafe behaviors but also to explain how such behaviors emerge in the system dynamics.

As an illustrative example, we consider a safety property related to inter-vehicle distance. The property states “that the radar distance of a vehicle should never remain in the *critically_low* range for more than 20 consecutive time steps”. Since the analyzed traces are sampled every 0.1 seconds, this corresponds to a maximum duration of 2 seconds during which the distance may remain in a critical state.

The verification of this property on traces corresponding to attack scenarios shows that the property is violated. This indicates that the system can reach configurations in which the radar distance remains in the *critically_low* range for longer than the allowed duration, thus violating the defined safety constraint.

The result of the property verification obtained using the Concurrency Workbench (CWB) is reported below.

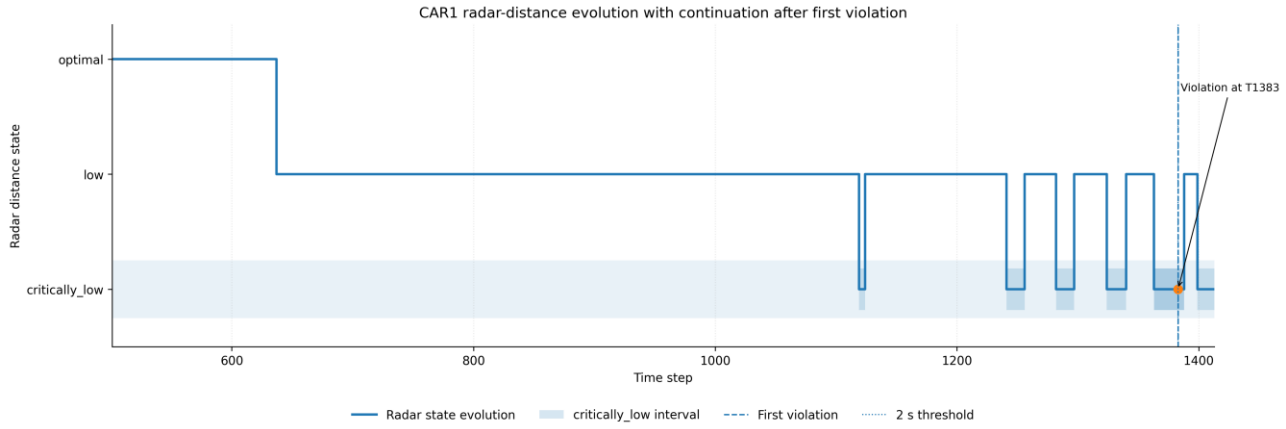
```
iitd-cwb>                    chk                    System                    is_d_qsafe_RC1
Invoking                  alternation-free                 model                      checker.
Building                                                                          automaton...
.........1000.........2000.........3000.........4000.........5000
.........
States:                                                                                5884
Transitions:                                                                           5883
Done                                  building                                   automaton.
FALSE,       the       agent       does       not       satisfy       the       formula.
Execution time (user,system,gc,real):(0.984,0.021,0.138,1.005)
```

The model checker determines that the formula is not satisfied by the analyzed system, confirming that the safety property is violated in the considered execution traces.

To further confirm that this behavior is reachable in the system model, we also verified a complementary property that checks whether there exists an execution trace in which the radar distance remains in the *critically_low* state for more than 20 consecutive time steps (see Appendix B). The verification result obtained with the Concurrency Workbench is reported below.

```
iitd-cwb> chk System exists_critically_low_sequence
Invoking alternation-free model checker.
Building automaton...
.........1000.........2000.........3000.........4000.........5000
.........
States: 5884
Transitions: 5883
Done building automaton.
TRUE, the agent satisfies the formula.
Execution time (user,system,gc,real):(0.672,0.005,0.038,0.676)
```

The result confirms that the system admits at least one execution trace in which the radar distance remains in the *critically_low* state for longer than the allowed duration. This complementary verification provides further evidence that the unsafe behavior detected in the analyzed trace corresponds to a reachable behavior of the system model. The execution trace illustrated in Figure below provides a concrete example of such behavior.



The above Figure shows the evolution of the radar-distance state for the considered vehicle over time. The vertical axis reports the discrete radar-distance states (*optimal*, *low*, and *critically_low*), while the horizontal axis represents the time steps of the execution trace. The shaded region corresponds to the *critically_low* distance range.

The blue curve represents the evolution of the radar-distance state along the trace. Several short transitions into the *critically_low* state can be observed during the execution, but these remain within the allowed duration. The orange circular marker indicates the first-time step at which the safety property is violated, i.e., when the radar distance has remained in the *critically_low* state for more

than the allowed number of consecutive time steps. The vertical dashed line indicates the time at which the safety property is violated.

The temporal relation between the attack activation and the property violation provides additional insight into the system behavior. In the considered simulation scenario, the attack is activated approximately 60 seconds after the beginning of the execution. The violation of the safety property is observed at time-step T1383, which corresponds to approximately 138.3 seconds in the simulation. This shows that the unsafe condition does not occur immediately after the attack activation, but rather after a delay of roughly 78 seconds. During this period, the system dynamics progressively evolve until the radar distance remains in the *critically_low* state for longer than the allowed duration. This behavior indicates that the attack gradually affects the platoon dynamics, eventually leading to the violation of the safety constraint.

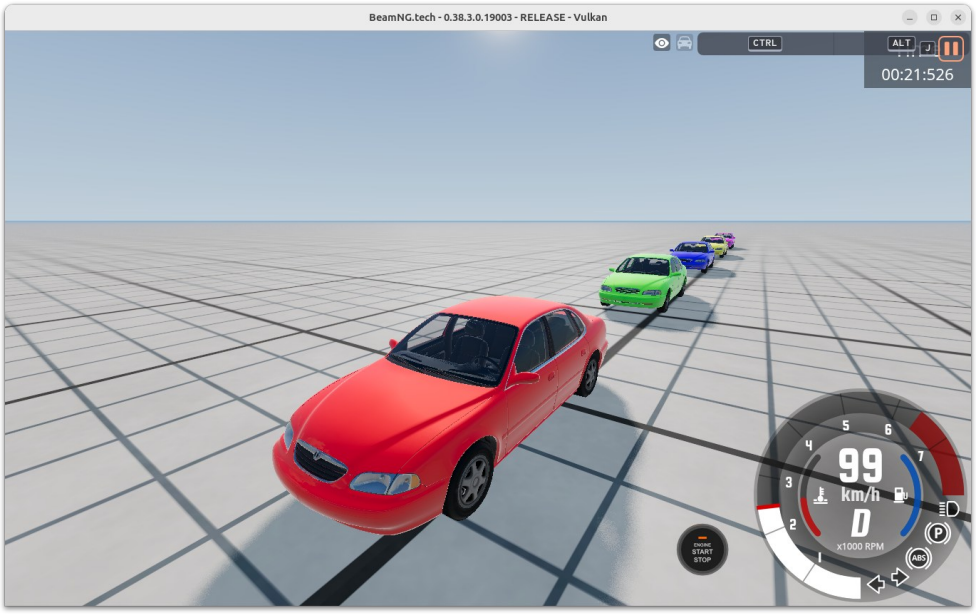
Although this example focuses on a specific property, the same reasoning can be applied to other behavioral constraints. More generally, the analysis of property violations provides a useful mechanism for explaining anomalous behaviors and understanding how attacks influence the dynamics of the cyber-physical system.

4. Graphical simulation of the platoon

BeamNG [BeamNG GmbH] is a simulation platform, distinguished by its custom soft-body physics engine and detailed vehicle modeling, which together enable the highly realistic replication of driving behavior and vehicle kinematics. Its adaptability, extensive modding system, and diverse scenarios make it an ideal environment for both the development of Advanced Driver-Assistance Systems (ADAS) through model-in-the-loop testing and the creation of immersive, customizable training applications for drivers.

The simulator provides a wide range of virtual sensors, such as radars, lidars, cameras etc.; that can be used to collect rich, multimodal data - including RGB images, depth information, object classes, and instance segmentation from the camera, highly customizable LiDAR point clouds, detailed IMU dynamics, ultrasonic proximity readings, and unique vehicle damage states - enabling comprehensive validation, sensor-fusion development, and system evaluation under programmable, scenario-based conditions without risk to real hardware.

An example of platoon simulation is shown in the following.



Our implementation of the systems is found at following links:

- https://github.com/ForeseenPRIN/platoon_controller_beam
- https://github.com/ForeseenPRIN/platoon_multimodel_beam
- https://github.com/scarburato/BeamNG-FMU (use the "modeldescfix" branch)

The BeamNG's FMU has been patched to make it work with the Maestro, the co-simulation orchestration engine of the INTO-CPS Association. A copy of BeamNG.tech has to be acquired to run these experiments.

4.1. Use in our co-simulation framework

The idea is to use the simulator's capabilities to provide a graphical visualization of the behavior of the platoon, as well as validating it against another simulator. To do so, we want to replace the less amount of FMUs as possible. In fact, this is one of the strengths of using a co-simulation framework; one may replace a part of the system with another leaving the other untouched. In our case, we just have to replace the FMU of the Car's Dynamics implemented in MATLAB Simulink with the official BeamNG FMU and a customized controller FMU.

We also wrote an auxiliary Python program that is used to bootstrap the simulator, create the platoon, and get it ready for communication with the FMUs.

4.2. Controller

Since so far in the project we have considered the case of a one dimension (1D) longitudinal platoon of vehicles, we have to do the same in this simulator as well. To this aim, we use the *grid map*, that is, an empty flat map, and we create the platoon along the positive X axis.

We make the platoon proceed without turning along the X axis to replicate the same driving scenario. To do so, we must control the steering of the car in such a way that the car's yawn remains to a constant angle, this is done by using a simple PID controller fine-tuned by hand.

Regarding acceleration control, the car is controlled by pressing the brake and/or throttle pedals in BeamNG, unlike the car's model used in MATLAB. The platoon is controlled by the desired acceleration signal generated by the CACC controllers; thus, we need a way to transform such a value into two values bounded between 0 and 1, one for the brake and one for the throttle. There are several ways to do such a job, for instance in [P. Ioannou, 1994] they used sophisticated model that takes into account the behavior of the automatic transmission, the engine power vs torque curve etc... for the throttle control; and the brake master cylinder oil pressure etc... for the brake input.

For simplicity's sake we use, again, a simple PID controller to control the two pedals. Such an approach is not optimal and generates some oscillations and irregularities during operations, especially when the automatic transmission switches gear; but for the purpose of a simple demo, it's more than sufficient as the overall behavior of the platoon – with or without attack – is not greatly affected.

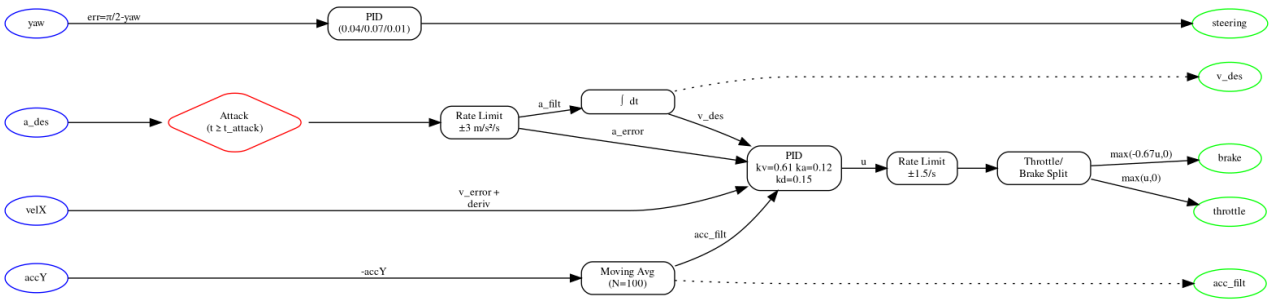


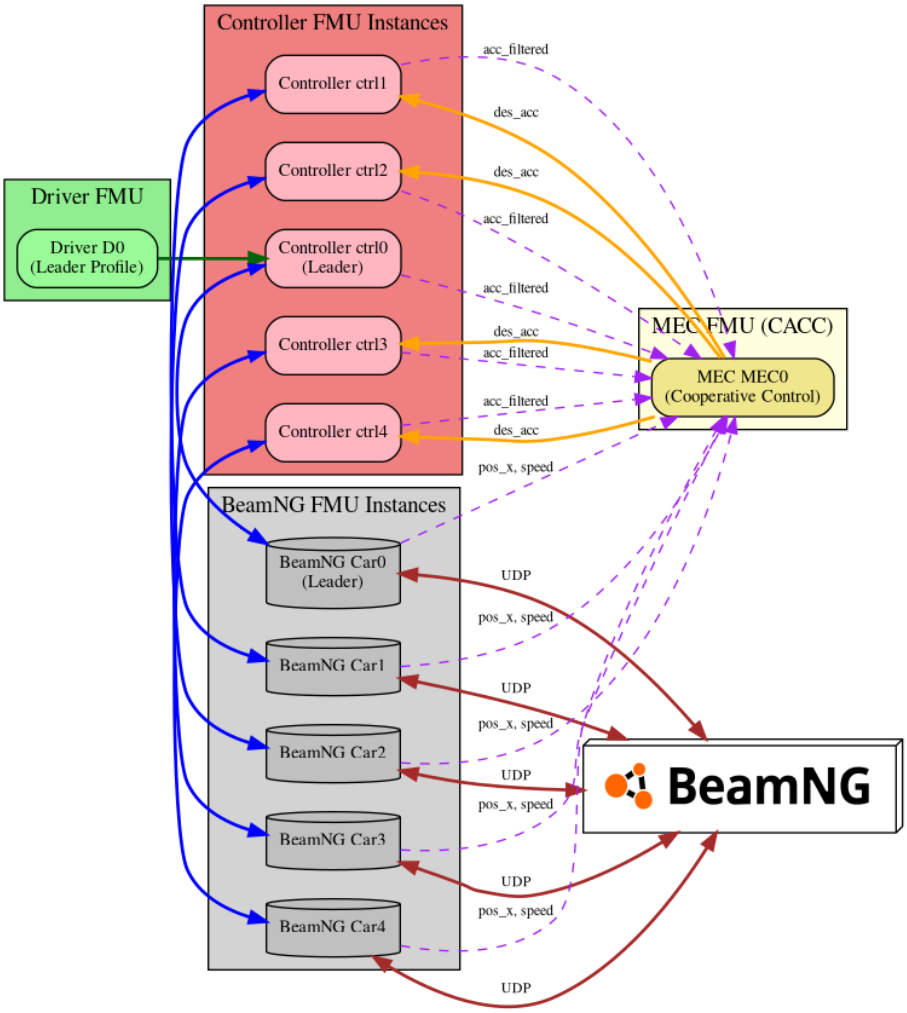
Figure above is a diagram of the control system used to control the car. Since the acceleration is quite noisy, because it's read through the simulator's virtual sensors, it passes through a simple moving average filter to reduce the noise.

In the controller, we also implemented the attacks on the actuators described in the previous deliverables.

4.3. Configuration and execution

The figure below shows the overall co-simulation architecture for *V2N communication*. The FMUs for the *V2V communication* scheme are connected in a similar way. However, as in the nominal case and as defined in previous deliverables, there are multiple FMU instances for the network.

We reuse the same FMU as in the nominal implementation for the *control law* and the *network communication*. Unlike the nominal implementation, the *car's controller* and the *car communication* are handled by two separate FMUs for the reasons discussed in previous sections.



The co-simulation is run with a simulation step of 5 milliseconds, in contrast to the 10 of the nominal case. This is due to the extra communication delay between the BeamNG FMUs and the car's controllers. It is possible to reduce the simulation step down to the simulator's resolution of the physics engine, that is, 0.5 milliseconds; but for our use-case 5 milliseconds were considered sufficient.

The simulation can be launched with the Makefile by running one or both of the following recipes, available on the project github:

- `build/modelV2N/outputs.csv`
- `build/modelV2N/outputs.csv`

4.4. Modeling of different road surfaces

Using the World Editor, it is possible to simulate the behavior of the platoon on different and heterogeneous road surfaces and conditions; in particular, the *terrain painter* may be used to change the surface of certain sections of the driving. For instance, we could simulate a stretch of icy motorway or a stretch of deformed road with potholes and bumps.

5. Conclusions

This deliverable presents the application of the formal-methods-based methodology developed during the project to detect attacks on connected autonomous vehicle systems, as applied to the platoon use case. The methodology is general and supported by automated tools. Analysis of abstract traces in the absence of attacks and under various attack scenarios is conducted by verifying predefined properties using a model checker. This process identifies a set of online tests derived from the system's formal properties, to support runtime monitoring activities aimed at detecting deviations from expected behaviour. Extracting counterexamples from the model checker to interpret violations of the expected behaviour improves the explainability of the anomalies detected.

Bibliography

[Cleaveland, 2000] Cleaveland, R., Li, T., & Sims, S. (2000). The Concurrency Workbench of the New Century: User's manual (Version 1.2). Dept. of Computer Science, SUNY at Stony Brook.

[Mil89] Milner, R. “Communication and concurrency”, ser. PHI Series in computer science. Prentice Hall, 1989.

[Koz83] D. Kozen, D. “Results on the propositional mu-calculus,” Theor. Comput. Sci., vol. 27, pp. 333–354, 1983. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(82)90125-6

[BeamNG GmbH] BeamNG GmbH, BeamNG.tech. 2025. [Online]. Available: https://www.beamng.tech/

[P. Ioannou, 1994] P. Ioannou and Z. Xu, “THROTTLE AND BRAKE CONTROL SYSTEMS FOR AUTOMATIC VEHICLE FOLLOWING,” I V H S Journal, vol. 1, no. 4, pp. 345–377, 1994, doi: 10.1080/10248079408903805.

Appendix A: Makefile

This appendix contains the GNU Makefile used to automate the full verification pipeline: for each simulation trace, it runs the discretization tool, invokes the CCS converter to produce the CWB-NC automata, and then executes the batch of μ-calculus queries via the CWB-NC wrapper. The final results are collected and merged into a single CSV file for analysis. Such a file is generated by the `merge.csv` recipe, which is also aliased as the `all` recipe.

The `PREFIXES` variable is the main configuration point of the Makefile: it lists the directories containing the simulation data for each batch scenario (e.g., no attack, attack on car 1's sensors, attack on the leader), and switching between the V2E and V2V datasets is achieved simply by pointing `PREFIXES` to the corresponding data directories, as shown by the commented-out alternatives at the top of the file.

As each trace is processed independently, the pipeline is parallel: running `make -jN` (where N is the desired number of parallel workers) exploits all available CPU cores and can dramatically reduce the total wall-clock time when processing large batches of simulation traces.

```
MAKEFLAGS += --no-builtin-rules
.PHONY: all clean clean-queries
.SUFFIXES:
.SECONDARY:

# Define all prefixes here
#PREFIXES := w_attack1 no_attack
#PREFIXES := $(shell find v2e_data/ -mindepth 1 -maxdepth 1 -type d 2>/dev/null)
#PREFIXES := v2e_data/Attack_First_Part-1 v2e_data/Attack_First_Part-2
v2e_data/Attack_Leader v2e_data/No_Attack
PREFIXES := v2v_data/Attack_First_Part-1 v2v_data/Attack_First_Part-2
v2v_data/Attack_Leader v2v_data/No_Attack

# Generate all output.csv targets for all prefixes
# i.e. all %/ouput.csv files
ALL_OUTPUTS := $(foreach prefix,$(PREFIXES),$(addsuffix /output.csv,$(shell find
$(prefix)/ -mindepth 1 -maxdepth 1 -type d 2>/dev/null)))
```

```
# Partitioning for merge
CHUNK_SIZE := 500
CHUNK_STARTS := $(shell seq 1 $(CHUNK_SIZE) $(words $(ALL_OUTPUTS)))
CHUNKS       := $(patsubst %, chunk_%.tmp.csv, $(CHUNK_STARTS))

all: merge.csv

merge.csv: $(CHUNKS)
@echo "Performing final stack of $(words $(CHUNKS)) chunks..."
csvstack $^ > $@

# Dynamic Rule Generation
# This 'foreach' loop creates a rule for every chunk start index.
# It uses $(wordlist) to grab exactly a slice of the file list.
define CHUNK_RULE
chunk_$(1).tmp.csv: $(wordlist $(1), $(shell echo '$(1) + $(CHUNK_SIZE) - 1' | bc),
$(ALL_OUTPUTS))
@echo "Merging chunk starting at $(1)..."
@csvstack $$^ > $$@
endef

$(foreach i,$(CHUNK_STARTS),$(eval $(call CHUNK_RULE,$(i))))

%/output.csv: %/merge.ccs prop_dist.mu queries.csv %/config.mm.json
python ../scripts/cwb_wrapper/cwb_wrapper.py $< prop_dist.mu --queries queries.csv
--output $@.tmp --run-name "$(@D)" --batch-name "$(word 2,$(subst /, ,$@))" 2>
/dev/null

@attack=$$(jq -r '.parameters | [."{Car1}.CarInstance_1.attack",
."{F1}.car1.attack", ."{Car2}.CarInstance_2.attack", ."{F2}.car2.attack",
."{Leader}.LeaderInstance.attack", ."{L}.li.attack"] | map(select(. != null)) |
map(select(. != 0)) | if length > 0 then .[0] else 0 end' "$(@D)/config.mm.json");
\
head -n 1 $@.tmp | sed "s/^/attack,/" > $@; \
tail -n +2 $@.tmp | sed "s/^/$$attack,/" >> $@; \
rm $@.tmp

%/merge.ccs: %/results.csv
python ../scripts/discretization/discretization_with_mode_bounds.py $< -o
$(@D)/labeled.csv
python ../scripts/ccs-converters/configurable/ccs_converter_sync_step_conf.py \
"$(@D)/labeled.csv" \
"$(@D)/ccs_output" \
```

```
--step 10 --skip-first 301
cat "$(@D)/ccs_output/LEADER.ccs" "$(@D)/ccs_output/CAR1.ccs"
"$(@D)/ccs_output/CAR2.ccs" > "$(@D)/merge.ccs"
#     printf \
#           "\n\n\nproc Env = 'sink0.'sink1.'sink2.'go0.'go1.'go2.Env\nproc System
= (LEADER_T301 | CAR1_T301 | CAR2_T301 | Env)\{sink0, sink1, sink2, go0, go1,
go2}\n" >> \
#           "$(@D)/merge.ccs"
printf \
"\n\n\nproc Env = 'sink0.'go0.'sink1.'go1.'sink2.'go2.Env\nproc System = (LEAD-
ER_T301 | CAR1_T301 | CAR2_T301 | Env)\{sink0, sink1, sink2, go0, go1, go2}\n" >> \
"$(@D)/merge.ccs"


clean-queries:
-rm merge.csv
-rm *.tmp.csv

-@for prefix in $(PREFIXES); do \
find $$prefix/ -name "output.csv" -exec rm "{}" \; ; \
done

clean: clean-queries
-@for prefix in $(PREFIXES); do \
find $$prefix/ -name "*.ccs" -exec rm "{}" \; ; \
done
```

Note how we merge the final result file “merge.csv” in chunks, it is done to avoid an issue with dash’s maximum number of command line arguments, as there are so many intermediate output.csv files that we exceed it!

MISSIONE 4
ISTRUZIONE
RICERCA

Appendix B: Formal Property Used for Behavior Explanation

This appendix reports on the formal specification of the property used to support the behavioral explanation discussed in Section 3.4 Explainability.

The property checks whether the system model admits executions in which the radar distance of a vehicle, i.e., CAR1, remains in the critically_low state for more than 20 consecutive time steps. The verification of this property confirms that the unsafe behavior observed in the analyzed trace corresponds to a reachable behavior of the system model.

Property used in the verification:

```
prop exists_critically_low_sequence =
min X = (<-> X \/ (
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low><-><-><-><-><-><-><-><-><-><-><-><-><-><->
    <car1_RADAR_DIST_critically_low> tt
))
```