Noname manuscript No.
(will be inserted by the editor)

Statistical Model Checking of a Dynamic Vehicle Platoon

Cinzia Bernardeschi · Adriano Fagiolini · Giuseppe Lettieri · Dario Pagani · Federico Rossi

Received: date / Accepted: date

Abstract In automotive engineering, vehicle platooning is a proposed method for improving convoy movements by coordinating them to increase the safety and efficiency of transportation systems. The complexity and stochastic nature of platooning systems provide difficulties for traditional model-checking techniques. However, Statistical Model Checking (SMC) offers a solution by using statistical inference to probabilistically evaluate system features. This paper shows that Uppaal SMC offers a robust framework for assessing platooning systems across various operational scenarios, combining statistical analysis and formal verification methodologies. The behaviour of the platoon is modelled stochastically to cover a wide range of driving scenarios and road surfaces. Using SMC, it has been possible to gauge the safety and functionality of the platoon by estimating the probability of several properties.

Keywords Cyber-physical systems · Statistical Model Checking · Vehicle Dynamics · Vehicle platooning

Cinzia Bernardeschi
University of Pisa, Pisa, Italy
E-mail: cinzia.bernardeschi@unipi.it (corresponding author)

Adriano Fagiolini
University of Palermo, Palermo, Italy
E-mail: adriano.fagiolini@unipa.it

Giuseppe Lettieri
University of Pisa, Pisa, Italy
E-mail: giuseppe.lettieri@unipi.it

Dario Pagani
University of Pisa, Pisa, Italy
E-mail: dario.pagani@ing.unipi.it

Federico Rossi
University of Pisa, Pisa, Italy
E-mail: federico.rossi@ing.unipi.it

1 Introduction

Vehicle platooning has emerged as a promising area of automotive engineering [24,25,32] in the pursuit of safer and more efficient transportation systems. Platooning involves a convoy of vehicles moving in tandem and coordinating their movements. This concept has the potential to transform the way roads are used for transportation: it uses modern communication and control systems to improve traffic flow, minimize fuel consumption and lessen the environmental effect of individual cars [22,23].

Due to the complex interaction of several nearby operating vehicles, ensuring system correctness under a variety of operating circumstances requires strict verification and validation procedures. Timed automata [1] offer a formal framework for methodically examining the temporal development of system behaviors and confirming compliance with the intended timing specifications in the context of model checking. Model-checking techniques [10,11] can efficiently traverse the state space of time automata to confirm the meeting of temporal features, such as deadlines, synchronization, and temporal ordering of events, by embedding timing constraints as part of the system model. Because accurate timing coordination is critical to the safe and effective operation of vehicle platooning systems, timed automata are also a vital formalism for utilizing model-checking techniques in the study and validation of these systems.

Although classical model-checking methods are highly effective in the analysis of systems, they are frequently not suitable for vehicle platooning, which is characterized by complex interactions and stochasticity. As a result, a method that can take into account the inherent unpredictability and variability of platoon-

ing systems is needed. This need is met by statistical model checking (SMC) [21], which uses statistical inference to evaluate system attributes probabilistically. This allows for the evaluation of vehicle platooning systems in a range of circumstances and uncertainties [2, 17]. Through the combination of statistical analysis's adaptability and formal verification's accuracy, SMC provides a potent framework for guaranteeing the security and dependability of platooning systems in practical applications.

This paper presents an investigation into the application of statistical model checking for vehicle platooning, addressing critical aspects of system design, analysis, and validation. The Uppaal modelling framework [4] is applied, which, integrating probabilistic reasoning with formal verification techniques, provides means for assessing the safety and reliability of platooning systems across a spectrum of operational conditions.

Contributions. The contributions of this work are:

- We embedded physical vehicle and control dynamics all within the same Uppaal model-based environment for Statistical Model Checking;
- We integrated the platoon application behavior with the rest of theUppaal model;
- We developed a system model that can be easily extended to expand the details of the model, such as increase the fidelity in the vehicle dynamics, or model the inter-vehicle communication by adding network delays, packet drops etc.

A preliminar version of this work appeared in [6]. The current work adds the torque and the wheels' slip to the vehicle dynamics model. These play a decisive role in enabling more precise control of the vehicle itself, and enables the study of system across a range of different road conditions.

The complete source code of the Uppaal project is available at `https://github.com/unipi-dii-uppaal/dynamic_vehicle_platoon`.

The paper is structured as follows: Section 2 deepens on the technologies and concepts used in this work; Section 3 illustrates related work and topics; Section 4 focuses on the modeling of the dynamic vehicle platoon; Section 5 shows the simulation and probabilistic verification steps, under different road surface conditions; Section 6 states the conclusions and possible future works.

2 Background

2.1 Formal methods and model checking

Formal methods in computer science refer to analytical techniques that have a strong mathematical foundation. Certain properties of a system can be guaranteed using formal approaches. In particular, Model checking [8,9] is an automatic method for the verification of reactive systems. The reactive system is represented as a state-transition graph, and specifications of properties are given in a propositional temporal logic. The state-transition graph is automatically searched to see if it satisfies the properties using an effective search process. Compared to theorem-proving verification techniques [31], model checking has several advantages. The process is highly automatic, which is crucial. A high-level representation of the model and the specification that needs to be verified is typically provided by the user. The model checker will either provide a counterexample execution that demonstrates why the formula is not satisfied, or it will finish with the answer true, indicating that the model fits the specification. For the purpose of identifying minute mistakes in intricate reactive systems, the counterexamples are very crucial. State-space explosion [12] is an issue with model checking when the amount of the state space to be checked increases exponentially with the size of the model utilized. In many situations, this has precluded traditional model checking from being used.

Statistical model checking (SMC) [21] is a method that aims to address this issue by using statistical analysis rather than precise model analysis. Known by another name, Monte Carlo simulation, it works particularly effectively with stochastic and probabilistic models [19]. To solve issues that are outside the scope of traditional formal methodologies, SMC can be employed in communication systems, embedded automotive systems, and aeronautics. The technique entails running a sufficient number of independent simulations of the model's behavior to produce a prediction of the behavior that is statistically valid. A system's attributes are not completely guaranteed by the SMC technique. However, there is no limit to how much the results' confidence can be raised. More tests can be carried out if a higher level of assurance is required. The number of simulations rises sublinearly with model size and linearly with certainty. Discrete and nonlinear phenomena are also naturally included in the same model by SMC.

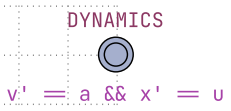

Fig. 1 Model of a car's kinematic

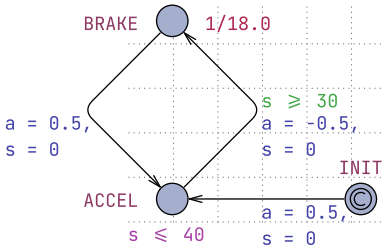

Fig. 2 A simple model of a driver

2.2 The Uppaal model checker for timed automata

One integrated tool environment for modeling and analyzing system behavior is Uppaal [4]. Uppaal is built on timed automaton (TA). It is possible to simulate and evaluate the timed behavior of different systems using networks of timed automata.

A timed automaton [1] is a directed graph over a set of clocks with a unique vertex known as the beginning position. We conventionally refer to vertices as locations. A reset set, an action, and a guard adorn an edge. If the source location is active and the guard evaluates to true, we say that an edge is enabled. A clock set is called a reset set. The idea is that each time the edge is gained, the clocks in the reset set are reset to zero. Keep in mind that following edges happens instantly and doesn't require any time. Lastly, invariants are used to mark the places. When an invariant's location is active, it is intuitive that it must evaluate to true. We can annotate the edges and the locations with costs and cost rates, respectively, to create a priced timed automaton.

A clock is a type of variable in Uppaal that is automatically incremented by one time unit every time the model evolves by one time unit.

Uppaal is particularly suitable for modeling hybrid automata [16], which are mathematical models that combine discrete (finite-state) and continuous (differential equation) aspects. Hybrid automata are useful for modeling systems in which analogue and digital processes interact. In this case verification of properties can be addressed within Uppaal SMC using priced TAs [5] and stochastic model checking. In particular, Uppaal SMC allows to modify the rate of a clock, when the automaton is in a certain state, by adding to its invariant a special notation resembling the Lagrangian prime derivative notation, that is `t' == expr`.

Listing 1 Definition of system quantities

```
clock x; // Car position
clock v; // Car speed
float a; // Car acceleration
```

For example, suppose we want to model a car being driven by a driver. The dynamic equations that express the position x, speed v and acceleration a are the following: $x' = v, v' = a$. In Uppaal we can define the state (x, v, a) with two clocks `x` and `v`, and the acceleration as a float variable `a`. The physics are modeled by imposing the prime derivative of the speed to the acceleration and the prime derivative of the position to the speed.

Figure 1 is the automaton that implements the physics of the car. Figure 2 is the automaton that implements the car's driver, it has a local clock variable `s` that is defined to control the timing.

The state `INIT` is the initial state the system finds itself at the beginning of time, as denoted by the two concentric circles. The **C** denotes that such a location is *committed*, that is this location must be exited immediately and Uppaal will force the system to transition to another location; in this case, the system will immediately transition to the `ACCEL` location. During the transition, the updates `a = 0.5, s = 0` are executed. The `ACCEL` location has only one transition going outwards, that leads to the `BRAKE` location. This transition has a guard `s >= 30`, which means that the system will be allowed to transition to the other location only when this condition is satisfied. The `ACCEL` has an invariant `s <= 40`, which allows the system to remain in that state only if the condition is satisfied. The resulting behavior is that the system will transition to the next state when $s \in [30, 40]$, with the exact value drawn from a uniform distribution; during the transition, `s` is reset and the acceleration `a` is set to -0.5.

The transition from `BRAKE` to `ACCEL`, instead, has no guard and the state `BRAKE` itself has no invariant. In Uppaal SMC we can also specify an exponential time distribution of transitions. Assigning λ to a location, the actual transition time will follow the probability distribution: $\lambda \cdot e^{-\lambda \cdot t}, \quad t \geq 0$. In our example, in the `BRAKE` location's properties, the rate of the exponential λ is set to `1/18.0`; in this case the time of transition to the other state is drawn from an exponential distribution with a mean value of 18.

Uppaal offers a *concrete simulator* to visually and numerically evaluate the simulation of the hybrid automata defined in the system. To simulate the behavior of such a system we introduce another automaton that enables a transition every time T. Figure 3 shows such a system, with $T = 1$: `t <= 1` is the initial location invari-

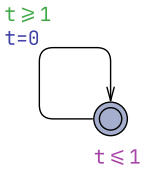

Fig. 3 Constant transition time of 1s

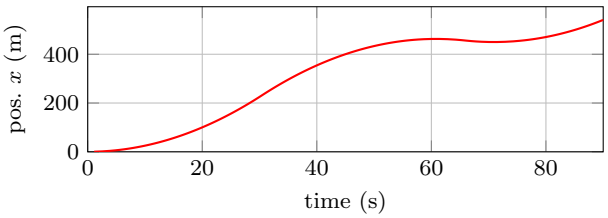

Fig. 4 Possible evolution of system over time. The value of the position x is plotted against time

ant, while `t >= 1` is the transition guard and $t = 0$ is the transition reset. This means that the transition can happen only when t equals 1 and then `t = 0` is the reset of t to 0. Hence, we obtain a transition that is enabled every second. Figure 4 shows a result of the simulation, showing the position of the car plotted against time.

2.3 Platooning and Cooperative Adaptive Cruise Control

Cooperative adaptive cruise control (CACC) [24] raises the bar for cruise control by allowing cars to react to speed changes made by the car in front of them quickly, by using a mix of sensors and vehicle-to-vehicle communication.

It has been demonstrated that the CACC algorithm satisfies the property of string stability [30,33], a fundamental prerequisite for guaranteeing the safety of a CACC system.

Intuitively, a platoon is said to be string stable if disturbances are not amplified when propagating through the platoon, from the leader going along the upstream direction.

For platoons to specifically attenuate either distance error, velocity, or acceleration along the upstream direction, string stability is a desirable quality. Indeed, if the string stability cannot be guaranteed, error signals will be amplified along upstream direction even if the closed-loop system is internal stable, eventually resulting in collision of two consecutive vehicles.

Let us consider a system where a platoon of vehicles is led by a *leader* car that is followed by a certain number of vehicles. The goal is to form a platoon of vehicles that move in unison, maintaining a fixed safety desired distance without colliding. We assume a

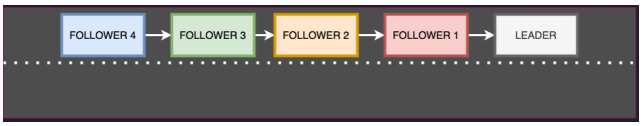

Fig. 5 A platoon formation

distributed CACC system installed locally on-board of vehicles and ideal and instantaneous vehicle to vehicle communications. Figure 5 shows a platoon formation.

The leader state can be identified by its longitudinal position, speed, and acceleration $\langle x_{leader}, \dot{x}_{leader}, \ddot{x}_{leader} \rangle$. The same holds for each of the following vehicle i, $\langle x_i, \dot{x}_i, \ddot{x}_i \rangle$.

A possible control equation that satisfies the desired properties of the CACC algorithm and the goals of the platoon can be obtained as follows. An exhaustive reference to possible alternative control laws for the CACC can be found in [30]. Suppose that vehicles are numbered from 1 to n, with the vehicle 1 being the closest to the leader. For each vehicle i, the desired acceleration $a^{(i)}_{ref}$, and the desired velocity $v^{(i)}_{ref}$, are:

$$a^{(i)}_{ref} = \ddot{x}_{i-1} + k_1 \cdot (\dot{x}_{i-1} - \dot{x}_i), \tag{1}$$

$$v^{(i)}_{ref} = \dot{x}_{i-1} + k_2 \cdot (x_{i-1} - x_i - d_{safe}), \tag{2}$$

where $\langle x_{i-1}, \dot{x}_{i-1}, \ddot{x}_{i-1} \rangle$ is the state of the preceding vehicle and d_{safe} is the desired fixed distance between vehicles in the platoon. The coefficients k_1, k_2 tune the control response.

3 Related work

The use of model checking for autonomous driving systems' verification and validation was investigated thoroughly in the literature.

In particular, in [18], the authors presented a multi-agent-based approach to formally verifying the coordination of multiple autonomous vehicles in convoy or platoon formations. This approach involves the use of a physical engine comprising a Matlab/Simulink model of the vehicles' physical properties. This model is then used to extract the characteristics of an abstract model implemented with timed automata, which specifies the algorithmic behaviour of platoon operations. The authors verified the safety and functional properties of the system.

A similar approach was explored in [13], where the authors combined a multi-agent system comprising a physical runtime and a model-checking environment for the formal verification of drone swarms.

In [20], model checking is used to improve the development of critical software for automated driving functions. The authors analyse the behaviour of the 'planner software component', which, based on the perceived flow of surrounding traffic, controls acceleration, braking and lane changes. These functions are demonstrated to ensure safe and compliant driving.

In [26], model checking is applied to provide the autonomous vehicle with a list of instructions for overtaking other vehicles. An autonomous vehicle and the surrounding traffic environment are modelled and analysed.

In [3], the authors applied Statistical Model Checking and Petri nets to a single autonomous vehicle immersed in a traffic jam scenario. In doing so, the authors were able to provide probability bounds to safety properties such as collision probability and the average distance before collision.

In [28], the authors rigorously modeled the collision detection and avoidance infrastructure of an autonomous vehicle and applied Statistical Model Checking to assess key performance indicators of said collision avoidance system.

In [15,27], the authors presented an approach for modeling physical quantities and their derivatives as clocks in Uppaal [4] and applied this approach to power electrical appliances.

With respect to previous approaches, we could fully exploit the Statistical Model Checking capabilities of Uppaal to statistically verify the whole system, from the vehicle dynamics and controls to the platoon behavior. Moreover, using SMC we could insert stochastic variability inside the platoon behavior without the need to explicitly model randomness in the model. Finally, the use of SMC also helped us to reduce the number of simulations required to reach the desired probability bounds for the assessment of safety and functional properties.

4 Platoon model in Uppaal

As described in Section 2, the platoon model fundamentally stems from the CACC control equations, which control the position, velocity and acceleration of vehicles. In this section we show how we modelled the cars' dynamics, implementing the torque and the wheels' slip.

Given a platoon of n vehicles $i = 1...n$ and a leader, the reference acceleration $a_{ref}^{(i)}$ and speed $v_{ref}^{(i)}$ for each of the vehicles are expressed by the equations (1) and (2), respectively.

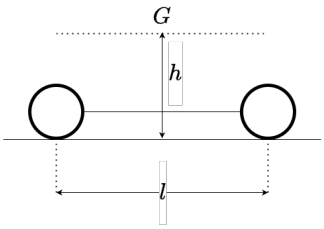

Fig. 6 Simple model of contact between wheels and road

4.1 Modeling the dynamics

Vehicles can be modeled as virtual wheels of mass m_i and radius R_i, each of them described by a state vector $\langle x_i, \dot{x_i}, \ddot{x_i}, \omega_i \rangle$, where ω is the angular speed of the wheel. Furthermore, the motion is controlled by the engine torque T_i that provides the longitudinal force $F_{lon}^{(i)}$. Whereas the L_i denotes the total loss of longitudinal force due to external conditions (e.g. aerodynamic drag). The torque applied to the vehicle's wheels is expressed by the following equation, where k is a gain parameter of the controller:

$$T_i = m_i \cdot R_i \cdot \left(a_{ref}^{(i)} - k \cdot (\dot{x_i} - v_{ref}^{(i)}) \right) + R_i \cdot L_i. \tag{3}$$

Hence we can write the dynamic equation that regulates the vehicles' motion as shown in (4)

$$m_i \cdot \ddot{x}_i = F_{lon}^{(i)} - L_i. \tag{4}$$

We're looking the physical value $\ddot{x} = (F_{\text{lon}}^{(i)} - L_i) \cdot m_i^{-1}$, that is the acceleration to impose on the car itself. We will use such a value to impose the prime derivative of position and speed accordingly, in a similar manner to what has been shown in the example of Section 2.2.

4.2 Road contact and friction model

To model the wheels and their friction with the road, we cannot assume perfect rolling. In this section we will introduce the formulae that describe the slipping, given the T_i and the current car's physical values. The longitudinal force can be expressed as follows:

$$F_{lon}^{(i)} = \mu_i(t) \cdot m_i g \cdot \frac{h_i}{l_i}, \tag{5}$$

where $\mu_i(t)$ models the friction with the road, $g = 9.81 \frac{m}{s^2}$ is the earth's gravity constant, h_i is the height of the vehicle center of mass G w.r.t to the wheels contact point with the road and l_i is the distance between the

Table 1 Road condition coefficients

Road condition	μ_1	μ_2	μ_3
Dry asphalt	1.28	23.99	0.52
Wet asphalt	0.86	33.82	0.35
Snow	0.19	94.13	0.06
Ice	0.05	306.39	0
Dry cobblestone	1.37	6.46	0.67
Wet cobblestone	0.4	33.71	0.12

contact points of the two wheels. Figure 6 illustrates the contact model.

Let us call J_i the wheels' inertia of the i-th car, then the angular acceleration $\dot{w}_i$ is controlled by the engine torque T_i as follows:

$$\dot{w}_i = \frac{1}{J_i} \cdot \left(T_i - R_i \cdot F_{lon}^{(i)} \right). \tag{6}$$

This is then implemented in Uppaal SMC by using the prime derivative notation. In this way, the physical value ω will be automatically updated as the integral of (6).

The slip ratio can be expressed as follows:

$$\sigma_i(t) = \frac{\omega_i \cdot R_i - \dot{x_i}}{\omega_i \cdot R_i}. \tag{7}$$

We can model the contact between the vehicle tires and the road. In particular, we model the friction function $\mu(t)$ using Burckhardt's model [14]:

$$\mu(t) = \mu_1(t) \cdot \left(1 - e^{-\mu_2(t) \cdot |\sigma(t)|}\right) - \mu_3(t) \cdot |\sigma(t)|, \tag{8}$$

where $\sigma(t)$ is the slip ratio defined in Equation (7) and μ_1, μ_2, μ_3 are the three friction coefficients, depending on road condition, as described in Table 1.

Figure 7 shows how the longitudinal force $F_{lon}^{(i)}$, simulated with a mass $m = 1500$kg and $h = 1$m, $l = 2$m. The five different curves correspond to the different longitudinal force responses when a constant torque T_i is applied ($\dot{\sigma}(t) = 0.01$). The higher the longitudinal force, the higher the linear acceleration is applied to the vehicle; indeed, on the one hand, with optimal conditions of dry asphalt (in yellow), the longitudinal force is higher. On the other hand, with an icy road (in violet), the longitudinal force is much smaller, resulting in a smaller linear acceleration applied to the vehicle.

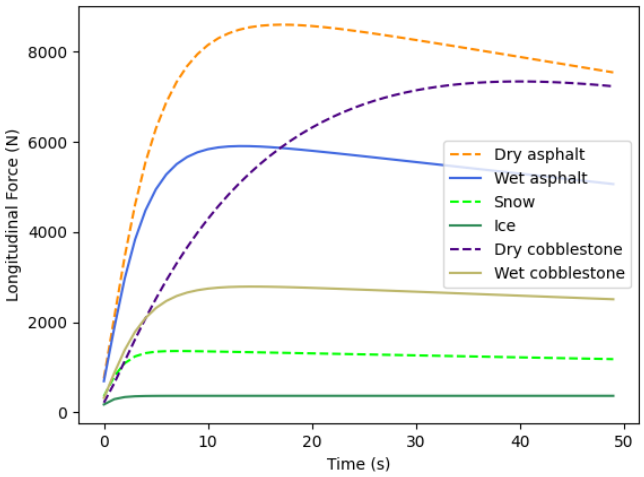

Fig. 7 Longitudinal force with different road conditions and constant torque

4.3 Uppaal models

We define three templates that model the components of the system:

- `LeaderDynamic`: models the leading vehicle behavior
- `FollowerDynamic`: models the follower vehicle behavior.
- `PlatoonCommand`: models the reference commands issued to the platoon leader
- `Simulation`: models the simulation progress for the concrete simulator inside Uppaal.

Listing 2 shows the composition of the system to form the platoon inside Uppaal. The physical quantities, i.e. the system state $\{\langle x_i, \dot{x}_i, \ddot{x}_i \rangle\}$ for $i = 0 \ldots$ `NUM_VEHICLES-1`, with `NUM_VEHICLES` the total number of vehicles (leader included), are stored as global variables:

```
clock positions[NUM_VEHICLES];
clock speeds[NUM_VEHICLES];
clock accelerations[NUM_VEHICLES];
```

and the values T_i, σ_i, ω_i are stored as local clock variables within each vehicle instance.

Index 0 is occupied by tòhe leader, while the others are occupied by the follower vehicles, corresponding to the platoon order. All these quantities are defined as clocks and their dynamic will be detailed in the following Subsections.

The reference acceleration `a_leader_ref` is passed by reference to the `Leader` from the `Command` process. Then, both `Leader` and `Follower1`, `Follower2`, ... vehicles are constructed by passing the references to the corresponding position, speed, and acceleration variables from the system declarations, i.e. the global variables.

To ease the follower construction, we used the `Follower(const int rank)` function to construct a `FollowerDynamic` process from its rank (lines 16-23 in Listing 2).

Listing 2 Composition of the platoon system

```
1  clock t;
2  const int NUM_VEHICLES = n + 1;
3
4  // Reference acceleration from the platoon
       command
5  double a_leader_ref = 0;
6
7  // Vehicles physical quantities
8  clock positions[NUM_VEHICLES];
9  clock speeds[NUM_VEHICLES];
10 clock accelerations[NUM_VEHICLES];
11
12 // Reference desired distance
13 double desired_distance = 20;
14
15
16 // Generator function for the followers dynamic
17 Follower(const int rank) =
       FollowerDynamic(
18     rank,           // Vehicle position in
       platoon
19     positions,      // Positions array
20     speeds,         // Speeds array,
21     accelerations, // Accelerations array
22     desired_distance
23 );
24
25 // Instance for the leader
26 Leader = LeaderDynamic(
27     t,
28     a_leader_ref,
29     positions[0],
30     speeds[0],
31     accelerations[0]
32 );
33
34 // Instances for the followers
35 Follower1 = Follower(1);
36 Follower2 = Follower(2);
37 ...
38 FollowerN-1 = Follower(N-1);
39
40 Command = PlatoonCommand(a_leader_ref,
41                          desired_distance);
42
43 Sim = Simulation(positions);
44
45 system Sim, Leader, Command, Follower1,
       Follower2, ..., FollowerN-1;
```

A follower in position i ($i \neq 0$) is constructed as

```
Follower1 = Follower(i);
```

The leader is constructed separately as the function `LeaderDynamic( t, ...)`, where t is the global time (lines 25-32 in Listing 2).

For readability, in the timed automata, the clocks: `positions[i]`, `speeds[i]` and `accelerations[i]` are named `x[i]`, `v[i]` and `a[i]`; respectively.

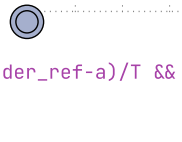

Fig. 8 Hybrid automaton that describes leader dynamics and behavior

The specification of the system is given by a network of automata which includes the Leader and the Command automaton, together with an instance of the Follower automaton for each follower (Listing 2, line 45).

4.4 Leader and vehicle dynamic

Figure 8 shows the hybrid automaton that describes the leader vehicle dynamics. The automaton has only one state, the initial one, with an invariant that follows a reference acceleration `a_leader_ref`, passed as a reference parameter to the process.

The reference is followed by a first-order control system with a time constant `T`, with the behavior described by the invariant `a' == (a_leader_ref - a)/T` (we assume $T = 2$ in our study, that simulates the actuation delay of the car's throttle). Then the rest of the leader's dynamics are modeled by `v' == a && x' == v`.

4.5 Platoon commands

The leader reference acceleration can be modelled with another automaton that simply behaves as an external actor issuing different trajectories to the entire platoon. Figure 9 shows the automaton that updates the reference acceleration `a_leader_ref`, read by the platoon leader.

The timed automaton defines a local clock `s`. In this particular example, from the initial location (committed), the first transition is enabled at `s==0` and updates the acceleration `a_leader_ref = 0.33`. After that, the local clock `s` periodically updates the reference acceleration. When `s` in the interval $[30, 40]$, the automaton moves to the next location updating the acceleration to `a_leader_ref = 0` (i.e. makes the platoon move with the same speed for a while) and resetting the clock to `s = 0`. In the next location, when `s` in the interval $[20, 30]$, the acceleration is updated to `a_leader_ref = -0.25` (i.e. the platoon decelerates) and finally, in the next location, when `s` in the interval $[10, 20]$, the transition to the `CYCLE` state sets back the acceleration to `a_leader_ref = 0.33`.

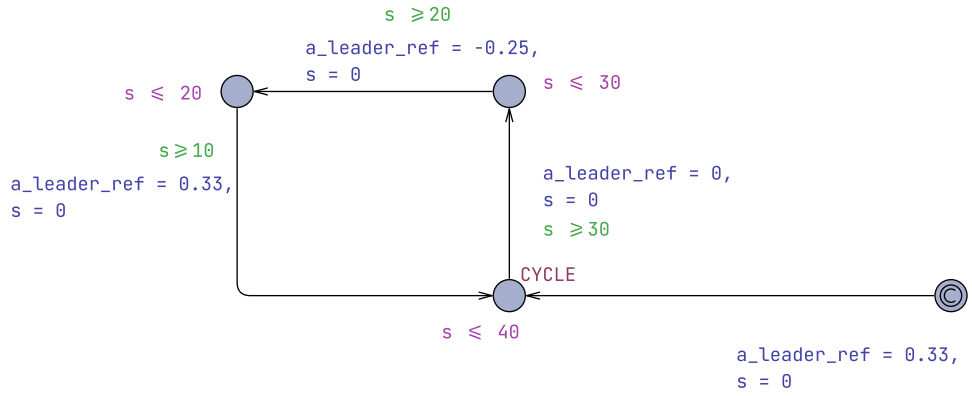

Fig. 9 Timed automaton that models commands issued to the platoon leader

4.6 Follower

A follower vehicle has two possible locations, depending on the role in the platoon:

- `NOT_JOINED`: the vehicle is moving independently and it is not part of the platoon already
- `JOINED`: the vehicle joined the platoon and it is following the vehicle in front

The automaton is depicted in Figure 10, the functions are shown in Listing 3.

Initially, no vehicle is inside the platoon. After a certain given time in the simulation, vehicles can start to join the platoon, transitioning into state `JOINED`. Each vehicle i joins the platoon after $3i$ seconds of waiting, thus the vehicles always join the platoon in order, one after the other, from front to back.

In the `JOINED` location, the follower dynamics is again expressed by Equation (4) and the reference acceleration is computed using the CACC control equation (see Equations (1), (2)). Therefore, the invariant for the derivative of the acceleration in the location `JOINED` is defined as
`a[i]' == (linear_acceleration() - a[i])/T1`,
where $T_1 = 0.01$, is the actuation delay of the follower car.

This template, called `FollowerDynamic`, is built with the constructor that has the following parameters, in order: `FollowerDynamic(int i,int next,clock& x[NUM_VEHICLES],clock& v[NUM_VEHICLES],clock& a[NUM_VEHICLES],double& d)`; where `i` is the index of the car, `next` is the index of the car in front, `x, v, a` are the physical values of all the cars in the platoon and `d` is the desired distance.

4.7 Joining and leaving the platoon

We want vehicles to be able to leave dynamically the platoon. In particular, we look for vehicles to leave the platoon from any position.

We introduce another location named `LEFT` to model a vehicle that left the platoon. For simplicity, we can assume that the leaving vehicle moves away from the same longitudinal axis of the platoon, making space for the following vehicle. Figure 11 visualize the operation of a vehicle (namely, `FOLLOWER 2`) leaving the platoon. To model this behavior, we slightly change the configuration of the platoon in Uppaal, adding an array of `broadcast chan` channels to let vehicles inside the platoon synchronize upon leaving. According to the Figure 11, the next of a vehicle (stored in the local variable `next`) is the vehicle in front of it. The transition to `LEFT` is controlled by the exponential rate 2.

We add two more parameters to the constructor's signature: `broadcast chan& platoon_left[NUM_VEHICLES], const bool leaving`, where the boolean constant `leaving` activates the leaving mechanism. Listing 4 shows the updates to the platoon configuration with `platoon_left[NUM_VEHICLES]` the array of broadcast channels.

Figure 12 shows the behavior of the hybrid automaton of the follower with the new location `LEFT`. When in `JOINED` there are two new transitions, let us assume we are in the template instance of the vehicle i:

- The transition with `platoon_left[next]?` is enabled whenever the vehicle in front of i leaves the platoon. During the transition the local variable `next` is updated as `next = next_after_leave` to assign the index of the new vehicle in front — that is the vehicle that was in front of the leaving one
- The transition with `leaving && s>=60` is enabled if the vehicle is marked as a `leaving` vehicle and when clock is `s>=60`. Upon transition, the vehicle signals

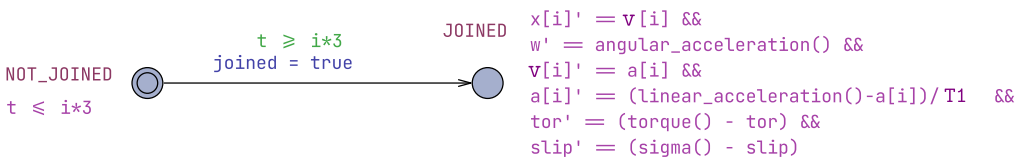

Fig. 10 Hybrid automaton that describes follower dynamics and behavior (JOIN the platoon)

Listing 3 An excerpt of the code of the follower

```
// Local clocks
clock w;       // angular velocity
clock slip;    // slip ratio
clock tor;     // engine torque

// next is the index of the car in front of car i

double desired_speed() { // Computes v_ref
    return v[next] + k2*(x[next] - x[i] -
    d);
}
double desired_acceleration() { // a_ref
    return a[next] + k1*(v[next] - v[i]);
}
double clamp(double in) {
    const double MAX_T = 1000;
    if(i < not_attacked_vehicle &&
    in>=MAX_T) return MAX_T;
    return in;
}
double torque() {
    double tor =
    ms*R*(desired_acceleration() - k *
    (v[i] - desired_speed())) +
    R*total_loss();
    return tor;
}
double sigma() {
    if(w<=0 || v[i] <= 0.3) return 0.0;
    return (w*R-v[i])/(w*R);
}
double mu() { // Friction model
    double abs_sigma = fabs(sigma());
    if (abs_sigma <=0) return 0.0001;
    return road_state[0]*
        (1-exp(-road_state[1]*abs_sigma))
        -road_state[2]*abs_sigma;
}
double longitudinal_force() {
    return mu()*m*g*(h/l);
}
double linear_acceleration() {
    return (1/m)*(longitudinal_force() -
    total_loss());
}
double angular_acceleration() {
    double tor = torque();
    if(USE_CLAMP)
        tor = clamp(tor);
    return (1/J)*(tor -
    R*longitudinal_force());
}
```

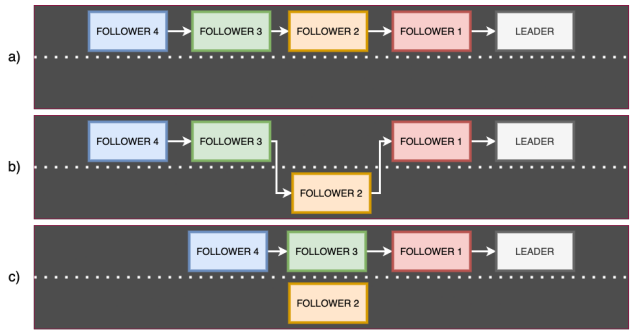

Fig. 11 Three-step evolution of the platoon when a vehicle leaves the formation

on its channel `platoon_left[i]!` the departure and goes into location `LEFT` where it does not follow the platoon dynamics anymore.

Listing 4 Platoon configuration and Follower function with platoon leave mechanics

```
// Channels for platoon leave synchronization
broadcast chan
    platoon_left[NUM_VEHICLES];

// Generator function for the follower's
    dynamic
Follower(const int rank,const bool
    leaving) = FollowerDynamic(
     rank,    // Vehicle position in platoon
     rank-1,  // Index of next vehicle
              // that is the veh. in front
     positions, // Positions array
     speeds,    // Speeds array,
     accelerations, // Accelerations array
     desired_distance,
     platoon_left,// Array of broadcast
    channels
     leaving // Vehicle marked for leaving?
);
```

The graphical representation of the model of the system with 1 Follower is shown in Figure 13. The automaton `Sim` is used to update, through the `accum_dif()`, the rolling averages of the following distances. It's used in Subsection 5.3.2.

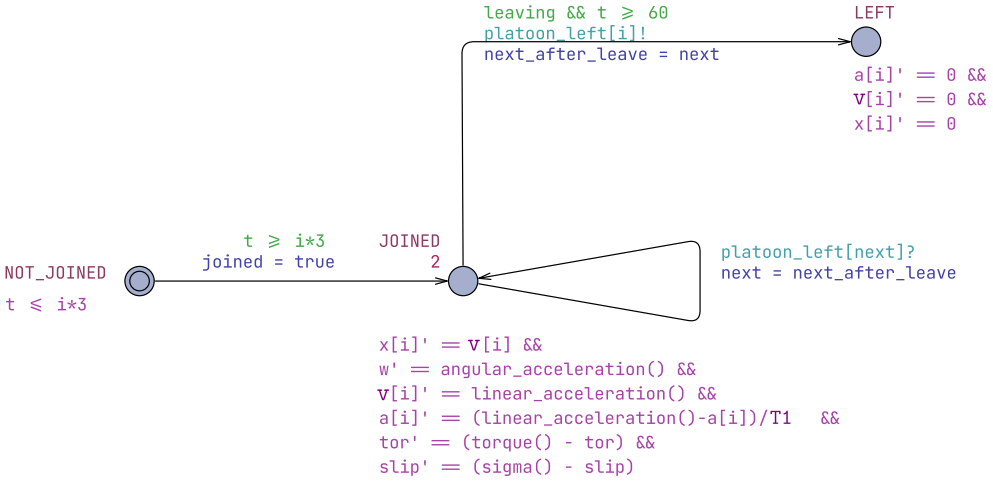

Fig. 12 Hybrid automaton that describes the follower vehicle dynamics and behavior (JOIN and LEAVE the platoon)

5 Simulation and probabilistic verification

In this Section, we proceed to first simulate the platoon and compare results with results from [29] to validate the Uppaal model. Then we analyse the platoon system under join and leave conditions to evaluate and validate the model visually. Finally, we apply SMC to provide probabilistic guarantees on the safety and functional properties of the system. We will refer to a platoon configuration with 4 vehicles, where the first one is the leader.

5.1 Uppaal model validation

In this subsection we provide a comparison between our work and another previous work [29] written by some of the authors. The other paper's implementation of a similar platooning scenario uses MATLAB SIMULINK to model the physics of the platoon and a control law written in C, all linked together via co-simulation using the Functional Mock-up Interface (FMI) [7] standard. In their driving scenario, the case of a car leaving the platoon was not modeled, so for the comparison's sake we disabled such a feature in this section. We also set the desired distance to 10 meters and the starting distance to the vehicle in front to 43 for Follower 1 and to 21 for the others; to recreate the same initial conditions.

A comparison between the two works, in the case of driving on dry asphalt, is shown in Figure 14. Qualitatively we can observe a good similarity between our evolution of the system and the one in the other work. In both works the following distance quickly converges to 10 meters.

In Table 2 we show a comparison of the average distance and standard deviation between each pair of cars in the case of dry and wet asphalt. The desired distance is set to 15 meters.

Table 2 Figures generated from two scenarios ran on Uppaal and Simulink

Scenario	**dist. b/w**	**Simulink** avg	dev	**Uppaal** avg	dev
Dry	0-1	14.53	1.03	14.12	0.87
	1-2	14.37	1.32	14.46	0.77
	2-3	14.46	1.16	14.89	1.22
Wet	0-1	14.51	1.07	14.14	0.85
	1-2	14.44	1.19	14.48	0.75
	2-3	14.40	1.24	14.90	1.20

In general, the discrepancy in the behaviors is due to slight differences between the way the physics is modeled in each solution.

5.2 Simulation: join and leave the platoon

An example of a simulation event trace can be seen in Figures 15 and 16. The figures show the event traces for two platoon configurations. In the first Figure, the followers join the platoon at fixed times 3, 6 and 9 seconds respectively. In the second Figure, `Follower2` leaves the platoon starting from `t>=60`.

In Figure 16, the red arrow shows the synchronization on the `platoon_left[2]` channel between the last two followers when the `Follower2` leaves the platoon. The synchronization is necessary to let `Follower3` know that it has to update its `next` variable to reflect the changed neighbour in the platoon. Note that this is not needed when the last follower in the chain leaves, and indeed there are no processes listening on its `platoon_-left` channel. This is not a problem, since send oper-

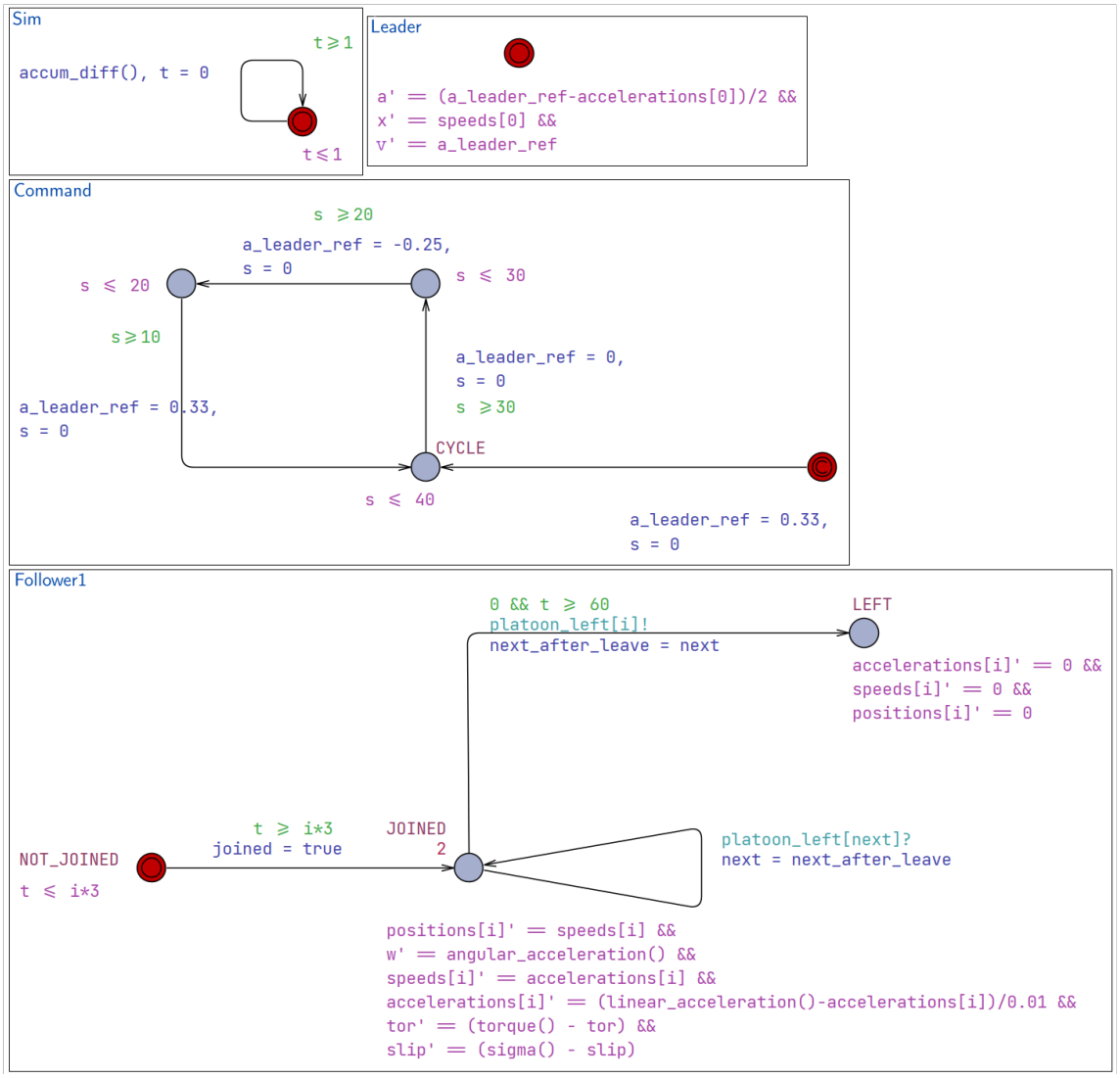

Fig. 13 Network of timed automata for the system with 1 follower and the leader

ations on broadcast channels are always enabled, even if there are no listeners.

5.3 Checking properties

In this subsection, we introduce a set of safety and functional properties to be proven with SMC. In UPPAAL a property that must always hold can be defined as

```
Pr[<=simulation_time] ( [] expr)
```

where `simulation_time` indicates how much time the system must be simulated for and `[]` indicates that `expr` must always hold. The term `expr` is a boolean expression that can contain either a global scope variable (e.g. `positions[0]`) or a process scope variable (e.g. `Follower1.joined`).

The overall system can be configured with several parameters, to explore different initial scenarios and requirements: i) Initial position and speed; ii) Desired safety distance between vehicles; iii) Control gain parameters; iv) Road condition coefficients; v) Leave time of the vehicles and whether vehicles can leave or not the platoon. We set up initial positions so that the vehicles are spaced by 20 m and initial speeds so that the vehicles start still with null speed. In all the experiments, the 1st follower leaves the platoon after 60 s, while the leader may accelerate by $0.33\,\mathrm{m/s^2}$ in the interval $[30\,s, 40\,s]$. The complete behavior of the leader is described in Figure 9.

We select 4 road conditions (dry asphalt, wet cobblestone, snowy and ice), 3 desired distances (20, 15 and 10 meters) and 4 maximum torque values (100, 200, 300

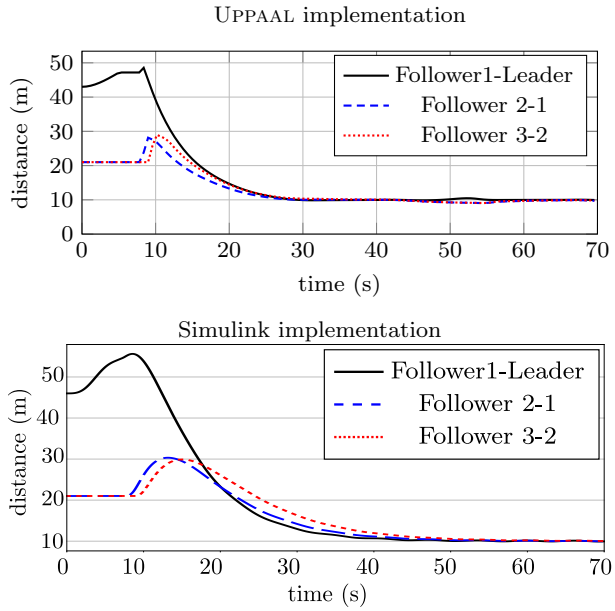

Fig. 14 Comparison of the distance between each pair of vehicles in the platoon. Desired distance set to 10 meters. Dry asphalt scenario. On the top: UPPAAL implementation; on the bottom: Simulink implementation. Bottom figure from [29], adapted for better readability

and 900 Newton-meters), then we test all combinations. The Confidence Interval is set to 97%.

5.3.1 Safety properties

Table 3 summarises the safety properties that we intend to prove using statistical model checking. Specifically, we want to assess the probability that the vehicle positions will not overlap during the platoon lifetime in the simulation, which is set to 300 s. Safety property S0 states that consecutive vehicles will never collide; assuming each vehicle is 4 m long, the distance between any two neighboring vehicles must always be at least 4 m. Safety properties S1, S2 and S3 assert the same property, but focus on a single pair of vehicles. This is useful in case S0 does not hold true, as it allows one to more precisely determine where in the platoon the property is violated.

The use of implication, `imply`, expresses a conditional propriety: if the follower has not left the platoon (`not f.LEFT`), then the positional constraint must hold.

The condition `positions[f.next] > positions[f.i] + 4` captures the required absence of collision between a follower vehicle `f` (indexed by `f.i`) and the vehicle it follows (indexed by `f.next`). These indices refer to entries in the positions' array, which maps vehicle identifiers to their physical values. The constant +4 accounts for the vehicles' length. For example, in property S1, the condition `positions[0] > positions[1] + 4` ensures that the leader (vehicle with index 0) is not rear-ended by Follower 1 (index 1), provided the latter has not left the platoon. This pattern is replicated across properties S2 and S3 for sub-

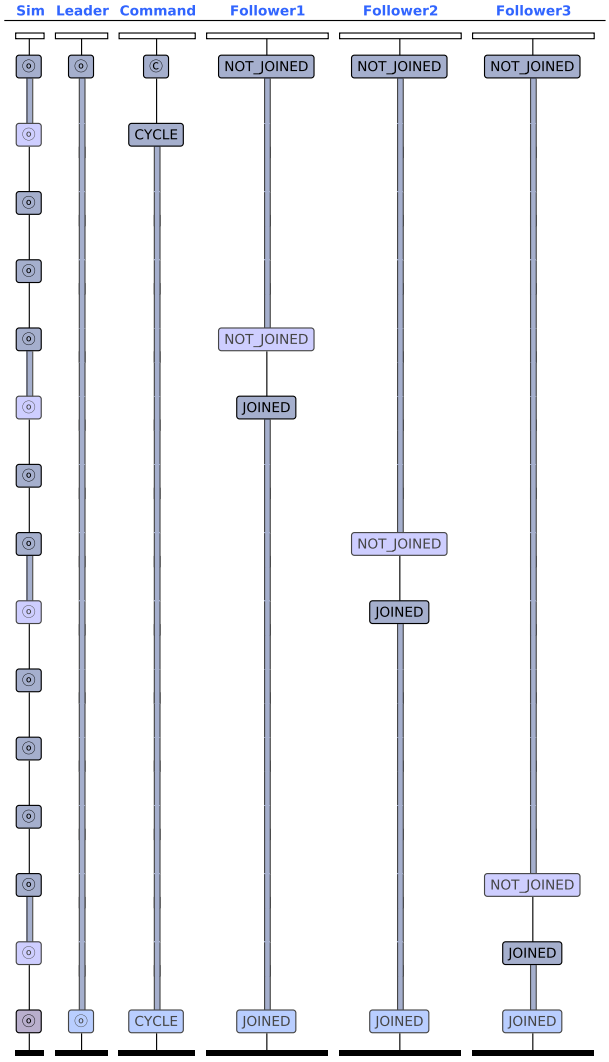

Fig. 15 Transitions of vehicle joining the platoon

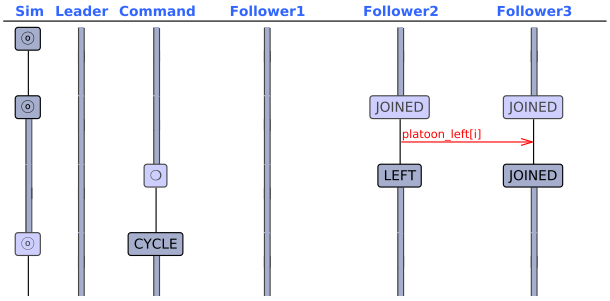

Fig. 16 Transitions of vehicle joining the platoon and `Follower2` leaving it, synchronising with `Follower3`

Table 3 Safety properties: absence of collisions in the platoon

ID	Name	Query
S0	No collisions between all pairs	`Pr[<=300] ([] forall(f : Follower) (not f.LEFT) imply positions[f.next] > positions[f.i] + 4)`
S1	No collision b/w Follower 1 and Leader	`Pr[<=300] ([] (not Follower1.LEFT) imply positions[0] > positions[1] + 4)`
S2	No collision b/w Follower 2 and the vehicle in front	`Pr[<=300] ([] (not Follower2.LEFT) imply positions[Follower2.next] > positions[Follower2.i] + 4)`
S3	No collision b/w Follower 3 and the vehicle in front	`Pr[<=300] ([] (not Follower3.LEFT) imply positions[Follower3.next] > positions[Follower3.i] + 4)`

Table 4 Desired distance 15 m, max torque 900 Nm, dry asphalt

Property	Probability	Successful runs/total
S0	[0.97, 1)	138/138
S1	[0.97, 1)	138/138
S2	[0.97, 1)	138/138
S3	[0.97, 1)	138/138

sequent follower-predecessor pairs, while S0 generalizes it to all followers using a universal quantifier.

In our case, S2 is equivalent to state `positions[1] > positions[2] + 4` until Follower 1 leaves the platoon at time 60 seconds, then it is equivalent to state `positions[0] > positions[2] + 4` as the `Follower2.next` has been changed from 1 to 0 during the leaving process.

Table 4 shows the results for the dry asphalt/ 15 m distance and a max torque of 900 Nm combination. We can see that the model checker has concluded that all the safety properties hold within the requested CI, since it has not detected any violation in 138 simulation runs. The same result holds also for all the other combinations of desired distance and max torque for the same road conditions.

Figure 17 illustrates a simulation run example. The figure plots the distance traveled by each vehicle over time during the first 120 seconds of the simulation. Note how `Follower1` leaves the platoon at 60 s, after which the other followers reduce their distance from the leader. The straight line in the first follower's plot after it leaves the platoon is not significant; it is just an artifact of how the `LEFT` state is implemented in Figure 12 and it does not imply that `Follower2` crashes into `Follower1`, since it is assumed that the follower leaves the platoon by moving into another lane (see Figure 11). However, this is not emulated or checked by the model.

Similar results are generated by both wet cobblestone and snowy road conditions, but we can note that,

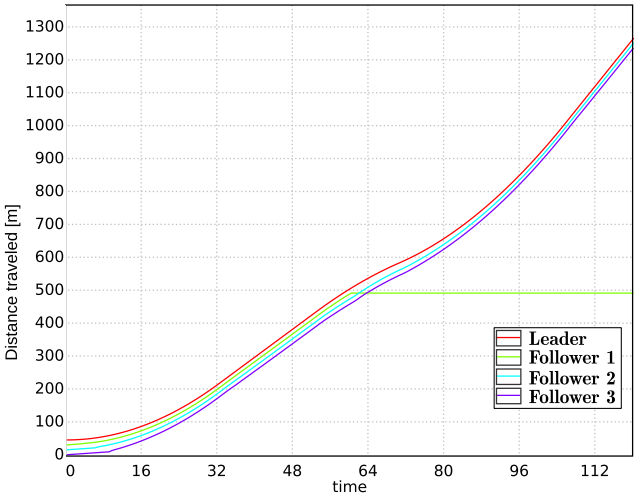

Fig. 17 Platoon positions for a desired distance of 15 m, 900 Nm max torque on dry asphalt

Table 5 Desired distance 15 m, max torque 300 Nm, ice conditions

Property	Probability	Successful runs/total
S0	[0.30, 0.36]	459/1271
S1	[0.97, 0.1)	138/138
S2	[0.95, 0.99]	239/244
S3	[0.30, 0.36)	450/1264

when the road conditions become worse, high torque limits may cause collisions, in accordance with physical intuition. For example, for the snowy road conditions, property S0 is satisfied with a probaility in the [.96, 1) range for a max torque of 300 Nm. This drops to [.94, 1) when the maximum torque is set to 900 Nm. Ice conditions introduce high instability for all desired distances and torque limits. For example, Table 5, shows that when we simulate ice conditions for a desired distance of 15 m with a torque limit of 300 Nm, there is an high chance (almost 2/3) that safety property S0 does not hold. Specifically, this is caused by the high likelihood

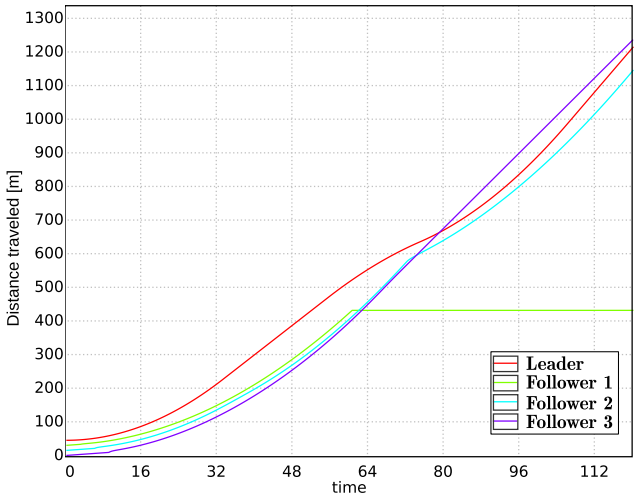

Fig. 18 Platoon positions for a desired distance of 15 m in the presence of ice

of a collision between `Follower2` and `Follower3`, as evidenced by the low probability of property S3.

Figure 18 shows an example run for this scenario. In this case, the leader accelerated around 20 s and then decelerated after 30 s; and the second follower was slow to match the desired distance, creating a large gap. After the second follower left the platoon, the remaining vehicles tried to catch up with the leader. When the third vehicle was about to reach the leader, it decreased its velocity. The fourth vehicle was too close at that instant and crashed into the third one (after this collision the simulation is no longer meaningful).

The model checker finds that a limit on the torque is effective in reducing the collision probability even in the presence of ice, but only for the lowest desired distance that we tested. Specifically, if the desired distance is set to 10 m, property S0 is found to hold (within the selected CI) if the torque limit is set to 100 Nm.

5.3.2 Functional properties

Table 6 details the properties that determine if the platoon positions will converge to the desired distance. The variables `Sim.distances[0]`, `Sim.distances[1]`, `Sim.distances[2]` contains the rolling average distance between pairs of vehicles in the platoon. For example, property F1 ensures that, after sufficient time has passed (estimated in 100 s), the average distance between the first and the second vehicles in the platoon will always be within 10% of the desired distance. If the desired distance is 20 m, `Sim.distances[0]` will be within 18 m and 22 m. Note that, in non-SMC Uppaal, these properties would have been written using the `A<>` operator, stating that a property eventually holds for each state, without the need to set a constant time after which we start checking the state property. Unfortunately, this operator is not available in Uppaal SMC.

In these tests, limits on torque can cause properties to fail, even in the most favorable cases. This is because the vehicles fail to react quickly enough to changes in the speed of the vehicle in front of them, creating transients in which the distance between vehicles falls outside the expected range. For instance, Table 7 shows that the model checker cannot find a successful run for any property, if the maximum torque is set to 200 Nm, even when the distance is set to 20 m on dry asphalt. Setting the maximum torque to 300 Nm satisfies properties F1 and F2, but not F3, which holds only with a probability between .16 and .21. Setting the torque limit to 900 Nm allows all properties to be satisfied in this scenario, as shown in Table 8.

The properties are not satisfied for any other combination of road conditions, desired distance, and torque limit. Unfavourable road conditions further reduce the effectiveness of the torque, which exacerbates the problems observed above.

More information can be found by checking the formulae in Table 9. These kind of formulas calculate the distribution of samples of the expressions following the `E[<=300; 500]` operator (the number after the semicolon indicates the number of runs and must be specified by the user). In the case of E1, the expression computes the maximum absolute value of the ratio between `Sim.distance[0]` and the desired distance after 100 seconds of simulated time. E2 computes the minimum of the same ratio. Similar formulas can be written for the other followers. The estimated means can be used to gauge the extent of the deviation from the desired distance. For instance, Uppaal reports an expected mean of $4.3 \pm .05$ for E1 and $1.3 \pm .004$ for E2 in the 20 m, dry asphalt, 100 Nm max torque scenario. This indicates that the deviation from the desired behavior is significant. Conversely, the results for 20 m dry asphalt and 200 Nm are very close to 1.04 for E1 and 1.05 for E2, indicating that the behavior may still be acceptable in this case. A limit of 100 Nm produces unacceptable results in all scenarios. A limit of 300 Nm, on the other hand, produces acceptable results for all desired distances in the dry asphalt scenario, and can also be accepted for a desired distance of 20 m on wet cobblestone; however, it is clearly insufficient for all other conditions.

6 Discussion and Conclusions

This work has emphasized the importance of statistical model checking (SMC) as a critical instrument for

Table 6 Functional properties: the correct inter-vehicular distance is maintained within a certain margin of error of ±10%

ID	Name	Query
F1	Correct distance b/w Follower 1 and Leader	`Pr[<=300] ([] t>= 100 imply ( Sim.distances[0] > desired_distance*0.90 && Sim.distances[0] < desired_distance*1.10))`
F2	Correct distance b/w Follower 2 and 1	`Pr[<=300] ([] t>= 100 imply ( Sim.distances[1] > desired_distance*0.90 && Sim.distances[1] < desired_distance*1.10))`
F3	Correct distance b/w Follower 3 and 2	`Pr[<=300] ([] t>= 100 imply ( Sim.distances[2] > desired_distance*0.90 && Sim.distances[2] < desired_distance*1.10))`

Table 7 Desired distance 20 m, max torque 100 Nm, on dry asphalt

Property	Probability	Successful runs/total
F1	$[0, .03)$	0/138
F2	$[0, .03)$	0/138
F3	$[0, .03)$	0/138

Table 8 Desired distance 20 m, max torque 900 Nm, on dry asphalt

Property	Probability	Successful runs/total
F1	$[0.97, 1)$	138/138
F2	$[0.97, 1)$	138/138
F3	$[0.97, 1)$	138/138

improving the dependability of vehicle platooning systems.

The model of the platoon includes complex physical aspects of the vehicle dynamic and SMC is shown to be a strong framework that can evaluate system characteristics in a probabilistic manner in a variety of operational scenarios.

The incorporation of SMC improves the comprehension of system behavior, and it also promotes well-informed choices during the design and implementation phases of the system. With respect to standard simulation approaches, in which a strategy to explore simulation runs and computation of statistical properties must be defined by the designer, SMC brought about the automatic computation of statistical properties of issues related to the uncertain nature of platooning systems.

The methodology can be applied in general to cyber-physical systems, where digital and physical components interact, because the modeling formalism supports derivatives and variability in the behavior, which are essential components such systems. The strength of this approach is the possibility of estimating automatically probabilistic properties on the system by providing logic formulae.

While the discussed model approximates the case of a platoon moving on a single dimension, the model could be modified to support movements in two or three dimensions, to allow, for instance, the study of the behavior of lane changing when a car is joining or leaving the platoon or the behavior of the platoon while traversing a curved and embanked section of road.

Such an extension can be implemented by adding new automata without touching the existing ones or by adding new locations and transitions to the current ones as well as by introducing new variables and clocks.

The probabilistic verification results underscore the critical role of road conditions and torque limitations in ensuring safety. While the model consistently satisfied collision-avoidance properties under dry asphalt conditions, icy roads introduced significant risks. These findings emphasize the necessity of adaptive control strategies tailored to environmental factors.

Further work will consider the application of SMC to evaluate the resilience of the platoon under different cyber-attacks scenarios, e.g. attacks to sensors/actuators or attack to communications; and to analyse safety for more realistic cooperative autonomous driving systems.

Acknowledgements This study received funding from the European Union - Next- GenerationEU - National Recovery and Resilience Plan (NRRP) - MISSION 4 COMPONENT 2, INVESTMENT N. 1.1, CALL PRIN 2022 PNRR D.D. 1409 14-09-2022 - (FORESEEN) CUP N.P2022WYAEW and from the Italian Ministry of University and Research (MUR) in the framework of the Crosslab and FoReLab projects (Departments of Excellence).

We'd like to thank the anonymous reviewers for the useful comments and suggestions.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science **126**(2), 183–235 (1994)
2. Barbier, M., Renzaglia, A., Quilbeuf, J., Rummelhard, L., Paigwar, A., Laugier, C., Legay, A., Ibañez-Guzmán, J., Simonin, O.: Validation of perception and decision-making systems for autonomous driving via statistical

Table 9 Expected values for maximum and minimum deviations from the desired distance of `Follower1`

ID	Name	Query
E1	Average max deviation	`E[<=300;500](max:(t>=100 ? 1 : 0.0)*fabs(Sim.distances[0])/desired_distance)`
E2	Average min deviation	`E[<=300;500](min:(t>=100 ? 1 : 1e800)*fabs(Sim.distances[0])/desired_distance)`

model checking. In: 2019 IEEE Intelligent Vehicles Symposium (IV), p. 252–259 (2019). DOI 10.1109/IVS.2019.8813793
3. Barbot, B., Bérard, B., Duplouy, Y., Haddad, S.: Statistical model-checking for autonomous vehicle safety validation. In: Conference SIA Simulation Numérique. Société des Ingénieurs de l'Automobile, Montigny-le-Bretonneux, France (2017)
4. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal, p. 200–236. Springer, Berlin, Heidelberg (2004). DOI 10.1007/978-3-540-30080-9_7
5. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Priced timed automata: Algorithms and applications. In: Formal Methods for Components and Objects: Third International Symposium, FMCO 2004, Leiden, The Netherlands, November 2–5, 2004, Revised Lectures 3, pp. 162–182. Springer (2005)
6. Bernardeschi, C., Lettieri, G., Rossi, F.: Statistical model checking of cooperative autonomous driving systems. In: T. Margaria, B. Steffen (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Rigorous Engineering of Collective Adaptive Systems, pp. 316–332. Springer Nature Switzerland, Cham (2025)
7. Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A.: Functional mockup interface 2.0: The standard for tool independent exchange of simulation models (2012). DOI 10.3384/ecp12076173
8. Browne, M.C., Clarke, E.M.: Sml-a high level language for the design and verification of finite state machines (1985)
9. Clarke, E.M.: Model checking. In: S. Ramesh, G. Sivakumar (eds.) Foundations of Software Technology and Theoretical Computer Science, p. 54–56. Springer, Berlin, Heidelberg (1997). DOI 10.1007/BFb0058022
10. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: D. Kozen (ed.) Logics of Programs, p. 52–71. Springer, Berlin, Heidelberg (1982). DOI 10.1007/BFb0025774
11. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986). DOI 10.1145/5397.5399
12. Clarke, E.M., Klieber, W., Nováček, M., Zuliani, P.: Model Checking and the State Explosion Problem, p. 1–30. Springer, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-35746-6_1
13. Desai, A., Dreossi, T., Seshia, S.A.: Combining model checking and runtime verification for safe robotics. In: 17th International Conference on Runtime Verification (RV), pp. 172–189 (2017)
14. Dousti, M., Baslamısli, S.C., Onder, E.T., Solmaz, S.: Design of a multiple-model switching controller for abs braking dynamics. Transactions of the Institute of Measurement and Control **37**(5), 582–595 (2015). DOI 10.1177/0142331214546522
15. Filipovikj, P., Mahmud, N., Marinescu, R., Seceleanu, C., Ljungkrantz, O., Lönn, H.: Simulink to uppaal statistical model checker: Analyzing automotive industrial systems. In: J. Fitzgerald, C. Heitmeyer, S. Gnesi, A. Philippou (eds.) FM 2016: Formal Methods, p. 748–756. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-48989-6_46
16. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science, pp. 278–292. IEEE (1996)
17. Huang, Z., Chu, D., Wu, C., He, Y.: Path planning and cooperative control for automated vehicle platoon using hybrid automata. IEEE Transactions on Intelligent Transportation Systems **20**(3), 959–974 (2019). DOI 10.1109/TITS.2018.2841967
18. Kamali, M., Dennis, L.A., McAree, O., Fisher, M., Veres, S.M.: Formal verification of autonomous vehicle platooning. Science of Computer Programming **148**, 88–106 (2017). DOI 10.1016/j.scico.2017.05.006
19. Katoen, J.P.: The probabilistic model checking landscape. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, p. 31–45. Association for Computing Machinery, New York, NY, USA (2016). DOI 10.1145/2933575.2934574
20. König, L., Heinzemann, C., Griggio, A., Klauck, M., Cimatti, A., Henze, F., Tonetta, S., Küperkoch, S., Fassbender, D., Hanselmann, M.: Towards safe autonomous driving: Model checking a behavior planner during development. In: B. Finkbeiner, L. Kovács (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 44–65. Springer Nature Switzerland, Cham (2024)
21. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, N. Tillmann (eds.) Runtime Verification, p. 122–135. Springer, Berlin, Heidelberg (2010). DOI 10.1007/978-3-642-16612-9_11
22. Li, S.E., Zheng, Y., Li, K., Wang, J.: An overview of vehicular platoon control under the four-component framework. In: 2015 IEEE Intelligent Vehicles Symposium (IV), pp. 286–291. IEEE (2015)
23. Liang, K.Y., Mårtensson, J., Johansson, K.H.: Heavy-duty vehicle platoon formation for fuel efficiency. IEEE Transactions on Intelligent Transportation Systems **17**(4), 1051–1061 (2016). DOI 10.1109/TITS.2015.2492243
24. Lu, X.Y., Hedrick, J., Drew, M.: Acc/cacc-control design, stability and robust performance. In: Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301), vol. 6, pp. 4327–4332 vol.6 (2002). DOI 10.1109/ACC.2002.1025325
25. Maiti, S., Winter, S., Kulik, L.: A conceptualization of vehicle platoons and platoon operations. Transportation Research Part C: Emerging Technologies **80**, 1–19 (2017)
26. Miller, A., Porr, B., Valkov, I., Fraser, D., Pagojus, D.: Model checking with memoisation for fast overtaking planning. Science of Computer Programming **244**, 103300 (2025)
27. Novak, M., Nyman, U.M., Dragicevic, T., Blaabjerg, F.: Statistical model checking for finite-set model predictive

control converters: A tutorial on modeling and performance verification. IEEE Industrial Electronics Magazine **13**(3), 6–15 (2019). DOI 10.1109/MIE.2019.2916232
28. Paigwar, A., Baranov, E., Renzaglia, A., Laugier, C., Legay, A.: Probabilistic collision risk estimation for autonomous driving: Validation via statistical model checking. In: 2020 IEEE Intelligent Vehicles Symposium (IV), p. 737–743 (2020). DOI 10.1109/IV47402.2020.9304821
29. Palmieri, M., Quadri, C., Fagiolini, A., Bernardeschi, C.: Co-simulated digital twin on the network edge: A vehicle platoon. Computer Communications **212**, 35–47 (2023). DOI 10.1016/j.comcom.2023.09.019
30. Rajamani, R.: Vehicle Dynamics and Control. Mechanical Engineering Series, Springer (2012)
31. Robinson, A., Voronkov, A.: Handbook of Automated Reasoning, vol. I. Elsevier, MIT Press. (2001)
32. Sheikholeslam, S., Desoer, C.A.: Longitudinal control of a platoon of vehicles. In: 1990 American Control Conference, pp. 291–296 (1990). DOI 10.23919/ACC.1990.4790743
33. Swaroop, D., Hedrick, J.: String stability of interconnected systems. In: Proceedings of 1995 American Control Conference - ACC'95, vol. 3, pp. 1806–1810 vol.3 (1995). DOI 10.1109/ACC.1995.531196