Statistical model checking of cooperative autonomous driving systems*

Cinzia Bernardeschi, Giuseppe Lettieri, and Federico Rossi

Department of Information Engineering, University of Pisa,
Via G. Caruso 16, Pisa (PI), Italy
cinzia.bernardeschi@unipi.it, giuseppe.lettieri@unipi.it,
federico.rossi@ing.unipi.it

Abstract. In automotive engineering, one of the ideas of vehicle cooperation is to improve convoy movements to increase the safety and efficiency of transportation systems. The complexity and stochastic character of such cooperative systems provide difficulties for traditional model-checking techniques. Statistical model checking (SMC) provides an answer by using statistical inference to evaluate system features probabilistically. In this work, we show how SMC offers a strong framework for assessing vehicle platooning systems in a range of operational scenarios by fusing statistical analysis and formal verification methodologies. Thanks to this approach, we managed to statistically prove a set of safety and functional properties of an autonomous vehicle platoon system.

Keywords: cyber-physical systems, statistical model checking, vehicle dynamics, vehicle platooning

1 Introduction

The development of vehicle platooning has emerged as a promising area in automotive engineering [1–3] in the pursuit of safer and more efficient transportation systems. The concept of platooning, which involves a convoy of vehicles moving in tandem and coordinating their movements, can completely transform the way that roads are used for transportation. Vehicle platooning is a strategy that attempts to improve traffic flow, minimize fuel consumption, and lessen the environmental effect of individual cars by utilizing modern communication and control systems [4, 5]. To ensure system correctness under a variety of operating circumstances, the complex interaction of several vehicles operating nearby necessitates strict verification and validation procedures.

* 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).

Model checking [6] is an approach used in formal verification to examine system designs for correctness against predetermined attributes. Model checking provides important insights into the robustness of a system by thoroughly examining the state space of the model to see whether it meets desired behavioural specifications. Timed automata (TA)[7] are a formalism for model checking real-time systems. TA are a type of formal model designed to reflect the temporal dynamics found in cyber-physical systems. By adding explicit timing restrictions and temporal logic specifications, these automata go beyond conventional finite-state machines and allow for the description and validation of complex timing-dependent characteristics. Timed automata offer a formal framework for methodically examining the temporal development of system behaviours and confirming compliance with intended timing specifications in the context of model checking. Model-checking techniques [8, 9] can efficiently traverse the state space of time automata to confirm the meeting of temporal features, such as deadlines, synchronisation, 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 cooperative autonomous driving 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 deterministic systems with finite state spaces, they are frequently not suitable for continuous-time systems with dynamic regulation by physical laws. In addition, with growing state complexity and system behaviour such techniques suffer from the so-called *state explosion* problem [10].

This need is met by SMC [11], which uses statistical inference to evaluate system attributes probabilistically. This allows for the assessment of cyber-physical systems across a range of circumstances and uncertainties [12, 13]. This approach extends the traditional timed automata with the rate of clocks given by general expression that involves other clocks, obtaining the physical behaviour described by ordinary differential equations. Such automata are called hybrid automata [14]. Furthermore, SMC provides a methodical way to assess complex systems with stochastic behaviour by combining formal verification methods with statistical analysis. In particular, SMC adds increased scalability and applicability to classic formal verification approaches.

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 formalism is flexible and allows the modelling of vehicles able to join and leave the platoon dynamically with little effort. By integrating probabilistic reasoning with formal verification techniques, we aim to provide means for assessing the safety of platooning systems across a spectrum of operational conditions.

The paper is structured as follows: Section 2 illustrates related work and this work main contribution; Section 3 deepens on the technologies and concepts used in this work; Section 4 focuses on the modelling of the dynamic vehicle platoon;

Section 5 shows the simulation and probabilistic verification steps; and Section 6 states the conclusions and possible future expansions.

2 Related work

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

In particular, in [15] the authors presented a multi-agent system to formally verify the coordination of multiple autonomous vehicles into convoy or platoon formations. The approach explores the use of a so-called physical engine that comprises a Matlab/Simulink physical model of autonomous vehicles. Such a model is used to extract the characteristics of an abstract model implemented with timed automata that models the algorithmic behaviour of platoon operations. The authors were able to verify the safety and functional properties of the system.

A similar approach was explored in [16] where the authors combined a multi-agent system comprised of a physical runtime and a model checking environment for formal verification for drone swarms.

In [17] the authors applied statistical model checking and Petri nets to a single autonomous vehicle immersed in a traffic jam scenario. By doing so, the authors were able to provide probability bounds to safety properties such as collision probability and average distance before collision.

In [18] 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 [19, 20] authors presented an approach to model physical quantities and their derivatives as clocks in UPPAAL [21] and applied it to power electrical appliances.

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 behaviour with the rest of the UPPAAL model.
- We developed a system model that can be easily extended to expand the details of the model (e.g. further complexity in the models of vehicle dynamics, interaction between vehicles and surrounding environment, and inter-vehicle communication)

With respect to previous approaches, we could fully exploit the statistical model-checking capabilities of UPPAAL to verify the whole system, from the vehicle dynamics and controls to the platoon behaviour. 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 required simulations to reach the desired probability bounds for the verification of safety and functional properties.

3 Background

This section illustrates basic knowledge of the technologies and concepts used in this work.

3.1 Statistical model checking

SMC [11] 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 [22].

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 behaviour to produce a prediction of the behaviour 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.

3.2 The UPPAAL model checker for timed automata

One integrated tool environment for modelling and analyzing system behaviour is UPPAAL [21]. UPPAAL is built on timed automaton (TA). It is possible to simulate and evaluate the timed behaviour of different systems using TA. However, as already pointed out when conventional numerical methods are applied, we end up with the state-space explosion problem, where the problem's size grows exponentially with the number of inputs. This can be addressed within Uppaal SMC using priced TAs [23] and SMC.

A timed automaton [7] 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 annotate the edges and the locations with costs and cost rates, respectively, to create a priced timed automaton.

UPPAAL is particularly suitable for modelling hybrid automata [24], where analogue physical processes and digital computer processes interact. For example, suppose we want to model a mass-damper-spring system controlled by an

external force. Figure 1 shows the system and the model of the system in UPPAAL. The dynamic equation that expresses the relation between position (x), speed ($\dot{x}$), and acceleration ($\ddot{x}$) of the mass (m) for the external force F_{ext} is

$$m\ddot{x} = F_{ext} - c\dot{x} - kx, \tag{1}$$

where k is the spring stiffness and λ is the damping coefficient.

In UPPAAL we can define the state $< x, \dot{x} >$ with two clocks x, v and express the differential relation using the derivative notation.

Listing 1.1: Definition of system quantities

```
clock x; // mass position
clock v; // mass speed
```

Assuming the definition of the system quantities in Listing 1.1, Figure 1 shows the modelling of such a system in UPPAAL with a network of two timed automata.

The quantities x, v are declared as `clock`, the notation x', v' expresses the derivatives of such quantities, F is the external force, passed-by-reference as an external parameter, l and k is the system coefficients c and k, respectively.

The `==` relation defines the invariant for the initial location of the first automaton: `x' == v && v' == (F-l*v-k*x)`.

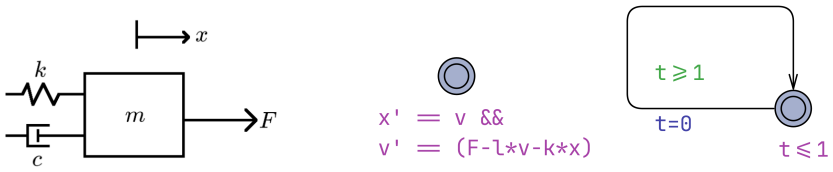


Fig. 1: The mass-spring-damper system and its model in UPPAAL

To simulate the behavior of such a system we introduce another automaton that enables a transition every time T. Such an automaton is shown on the right side of Figure 1: `t <= 1` is the initial location invariant while `t >= 1` is the transition guard and $t = 0$ is the transition reset. The transition can happen only when t equals 1 and then `t = 0` is the update to 0. Hence, we obtain a transition that is enabled every second. At each time T also the invariants of the first automaton are evaluated.

UPPAAL offers a *concrete simulator* to visually and numerically evaluate the simulation of the hybrid automata defined in the system. Figure 2 shows a result of the simulation, reporting the position of the mass over time when a constant force is applied to it.

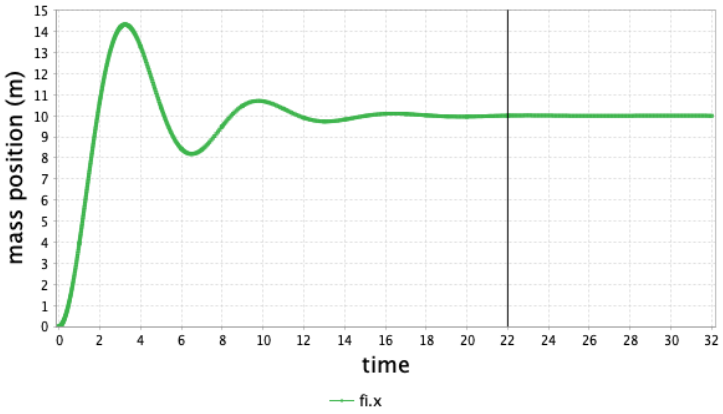


Fig. 2: Step-response plot of the position of the mass in the mass-spring-damper system.

3.3 Cooperative Adaptive Cruise Control and platooning

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 distance without colliding.

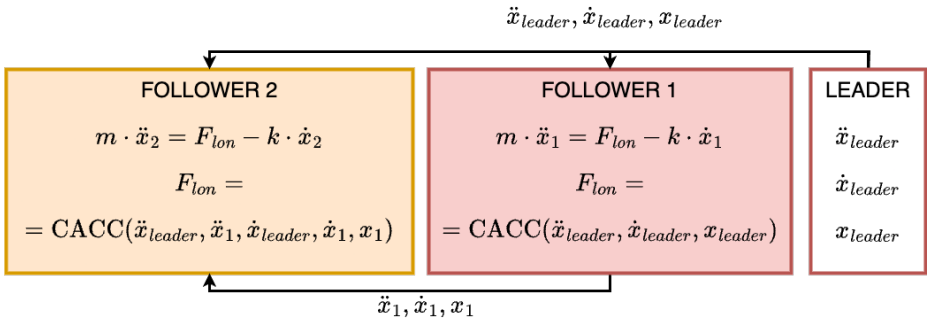


Fig. 3: Schematization of the platoon with 2 followers and 1 leader and vehicle-to-vehicle communications.

Cooperative adaptive cruise control (CACC) algorithm [1, 2] raises the bar for cruise control by allowing cars to react to the car in front of them in their lane by using a mix of sensors and vehicle-to-vehicle communication. Additionally, the CACC system can react to speed changes made by the car in front of you and by other cars that are farther away and out of your field of vision faster. It has been demonstrated that the CACC algorithm satisfies the property of string stability

[25], a fundamental prerequisite for guaranteeing the safety of the system. 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 the upstream direction even if the closed-loop system is internally stable, eventually resulting in the collision of two consecutive vehicles.

Modeling of a vehicle For simplicity, vehicles can be modelled as material points of mass m, each of them described by a state vector $\langle x, \dot{x}, \ddot{x} \rangle$. Aerodynamic drag interaction can be modelled with a coefficient k that multiplies the speed ($k \cdot \dot{x}_i$). Furthermore, the motion is controlled by the reference acceleration that provides the longitudinal force $F_{lon}^{(i)}$ (we assume a perfect slip ratio between the wheel and the road). Hence we can write the dynamic equation that regulates the vehicles' motion:

$$m \cdot \ddot{x}_i = F_{lon} - k \cdot \dot{x}_i. \tag{2}$$

We can then model the vehicle communication delay of the reference acceleration with a first-order control system. Let a be the vehicle acceleration ($a_i = \ddot{x}_i$), the communication delay behaviour can be expressed as follows, with τ being the time constant of the system (i.e. from control theory, the time it takes for the step response to rise to 63%):

$$\dot{a}_i = \frac{a_{ref}^{(i)} - a_i}{\tau}. \tag{3}$$

CACC control law The leader state can be identified by its longitudinal position, speed, and acceleration $\langle x_{leader}, \dot{x}_{leader}, \ddot{x}_{leader} \rangle$ and follow a possible predetermined path. The same holds for each of the following vehicle i, $\langle x_i, \dot{x}_i, \ddot{x}_i \rangle$. The control equation that satisfies the desired properties of the CACC algorithm is the follows. 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 $a_{ref}^{(i)}$ is:

$$a_{ref}^{(i)} = c_1 \cdot \ddot{x}_{leader} + (1 - c_1) \cdot \ddot{x}_{i-1} - k_1 \cdot (\dot{x}_i - \dot{x}_{leader}) - k_2 \cdot (x_i - x_{i-1} + d_{safe}), \tag{4}$$

where $\langle x_i, \dot{x}_i, \ddot{x}_i \rangle$ is the state of the preceding vehicle and d_{safe} is the desired fixed distance between vehicles in the platoon. The coefficients c_1, k_1, k_2 tune the control response. The value c_1 weights the acceleration value received by the leader and the front vehicle. The higher the value of c_1, the less sensible to fluctuations of the front vehicle is the control. The values $k1, k2$ model two gains on the control of speed and position, respectively.

Figure 3 shows the functional scheme of a platoon with a leader and two following vehicles. In the Figure, the CACC function implements the reference acceleration evaluation according to the CACC algorithm in Equation (4).

4 Platoon model in UPPAAL

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

- `LeaderDynamic`: models the leading vehicle behaviour.
- `FollowerDynamic`: models the follower vehicle behaviour.
- `PlatoonCommand`: models the reference commands issued to the platoon leader, i.e., the reference acceleration.
- `Simulation`: models the simulation progress for the concrete simulator inside UPPAAL.

The physical quantities, i.e. the system state $\{\langle x_i, \dot{x}_i, \ddot{x}_i \rangle\}$, $i = 0 \dots \mathrm{N} - 1$ are:

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

The leader occupies Index 0, while the follower vehicles occupy the others, 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 `leader_acceleration` is passed by reference to the `Leader`. and `Command` processes. Then, both `Leader` and `Follower1, Follower2, ...` vehicles are constructed passing by reference to the correspondent position, speed, and acceleration variable from the system declarations.

The leader is constructed as follows:

```
1 LeaderDynamic(
2     t,
3     leader_acceleration,
4     positions[0],
5     speeds[0],
6     accelerations[0]
7 );
```

To ease the construction of followers we used the *`Follower(const int rank )`* generator function to construct a `FollowerDynamic` process from its rank as in Listing 1.2. A follower in position i ($i \neq 0$) is constructed as

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

Listing 1.3 shows the composition of the system to form the platoon inside UPPAAL.

4.1 Leader dynamic

In the model, leader vehicle dynamics follows the exact behaviour expressed in Equation (2). Figure 4 shows the hybrid automaton that describes the leader vehicle dynamics. The automaton has only one state, the initial on, with an

Listing 1.2: Follower vehicle generator function

```
// Generator function for the follower dynamic
Follower(const int rank, const bool leaving, double& leave_time) =
FollowerDynamic(
    t,  // global clock time
    leader_acceleration,  // platoon leader acceleration
    speeds[0],  // platoon leader speed
    positions[rank-1], // Position of the front vehicle
    speeds[rank-1],    // Speed of the front vehicle,
    accelerations[rank-1], // Acceleration of the front vehicle
    positions[rank],  // Output variable for the vehicle position
    speeds[rank],     // Output variable for the vehicle speed
    accelerations[rank], // Output variable for the vehicle acceleration
);
```

```
acceleration' == (a_reference - acceleration)/T &&
speed' == (acceleration*tt-alpha*speed)/mass &&
position' == speed
```

Fig. 4: Hybrid automaton that describes leader dynamics and behaviour

invariant that follows a reference acceleration `a_reference`, 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 behaviour described by the invariant `acceleration' == (a_reference - acceleration)/T`. Then the rest of the dynamic interaction with the road is modelled by the invariant `speed' == acceleration - alpha*speed/mass`. Where `mass` is the vehicle mass and `alpha` is the dynamic friction coefficient with the road.

4.2 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 5 shows an example automaton that updates the reference acceleration `a_leader`, read by the platoon leader. In this particular example, from the initial location, the first transition is enabled at `t>=10` and updates the acceleration `a_leader = 0.5`. After that, another local clock `local_t` is used to periodically update the acceleration. When `t>=10` the automaton transitions to the next location updating the acceleration to `a_leader = 0` (i.e. makes the platoon move with the same speed for a while) and resetting the clock to `local_t = 0`. In the next location, when `t>=10`, the acceleration is updated to `a_leader = -0.5` (i.e. the platoon decelerates) and finally, in the next location, when `t>=5` the transition to the `CYCLE` state sets back the acceleration to `a_leader = 0.5`.

Listing 1.3: Composition of the platoon system

```
clock t;
const int NUM_VEHICLES = N;

// Reference acceleration from the platoon command
double leader_acceleration = 0;

// Vehicles physical quantities
clock positions[NUM_VEHICLES];
clock speeds[NUM_VEHICLES];
clock accelerations[NUM_VEHICLES];

// Generator function for the follower's dynamic
Follower(const int rank, const bool leaving, double& leave_time) =
    VehicleDynamic(
        t,  // global clock time
        accelerations[0],  // platoon leader acceleration
        speeds[0],  // platoon leader speed
        positions[rank-1], // Position of the front vehicle
        speeds[rank-1],    // Speed of the front vehicle,
        accelerations[rank-1], // Acceleration of the front vehicle
        positions[rank],  // Inout var for the vehicle position
        speeds[rank],     // Inout var for the vehicle speed
        accelerations[rank], // Inout var for the vehicle acceleration
    )

Leader = LeaderDynamic(
    t,
    leader_acceleration,
    positions[0],
    speeds[0],
    accelerations[0]
);
Follower1 = Follower(1);
Follower2 = Follower(2);
...
FollowerNm1 = Follower(N-1);


Command = PlatoonCommand(t, leader_acceleration);

Sim = Simulation(s);

system Sim, Leader, Command, Follower1, Follower2, ..., FollowerNm1;
```

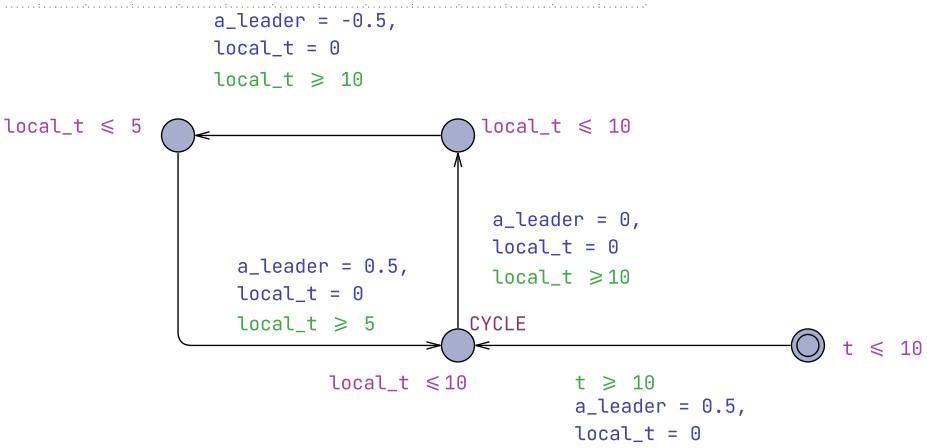


Fig. 5: Hybrid automaton that models commands issued to the platoon leader

4.3 Follower dynamic

We want to allow vehicles to join and leave the platoon. Two models for follower vehicles are defined.

Basic follower Figure 6 shows the basic follower dynamic and controls. The reference acceleration is computed using the CACC control equation (see Equation (4)); the rest of the dynamic is the same as the one described in Equation (2). Therefore the invariant for the derivative of the acceleration is defined as `acceleration' == ( eval_accel() - acceleration)/T`. The acceleration is evaluated by the function *`eval_accel()`* in Listing 1.4.

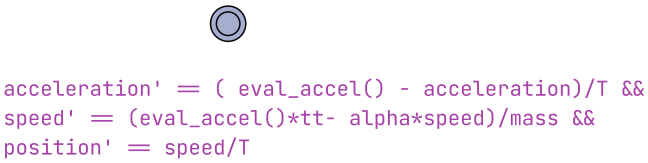


Fig. 6: Hybrid automaton that describes simple follower dynamics and behaviour

Joining and leaving the platoon In the case of join and leave, a follower vehicle has three possible locations as shown in Figure 7, depending on the role in the platoon:

– `NOT_JOINED`: the vehicle is moving independently and it is not part of the platoon already

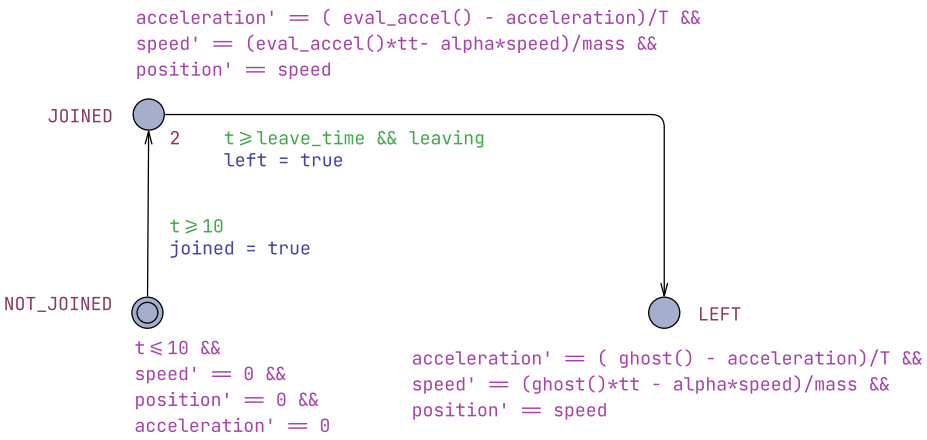


Fig. 7: Hybrid automaton that describes follower dynamics and behaviour with join and leave

Listing 1.4: Reference acceleration computation according to CACC Equation (4).

```
double eval_accel() {
    double accel = c*a_leader+(1-c)*a_front - K1*(speed-v_lead) -
     K2*(position-x_front+dsafe);
    if(speed <= 0)
        return fmax(0, accel);
    else
        return accel;
}
```

- JOINED: the vehicle joined the platoon and it is following the front vehicle according to the CACC Equation (4).
- LEFT: the vehicle abandoned the platoon and it is no longer part of the convoy (i.e. moved into another lane).

In all the locations, the follower dynamics is expressed again by Equation (2). When the follower is in the JOINED location the reference acceleration is computed using the CACC control equation (see Equation (4)); the rest of the dynamic is the same as the one described in Equation (2). We slightly modified the algorithm to not allow vehicles to go in reverse when closer than the safety distance but standing still.

We want vehicles to be able to join and leave dynamically the platoon. In particular, we look for vehicles to leave the platoon from any position and to join into the back of the platoon queue. If we look at Figure 7 the NOT_JOINED state serves this purpose. Initially, no vehicle is inside the platoon. After a certain given time in the simulation, vehicles can start to join the platoon, transitioning

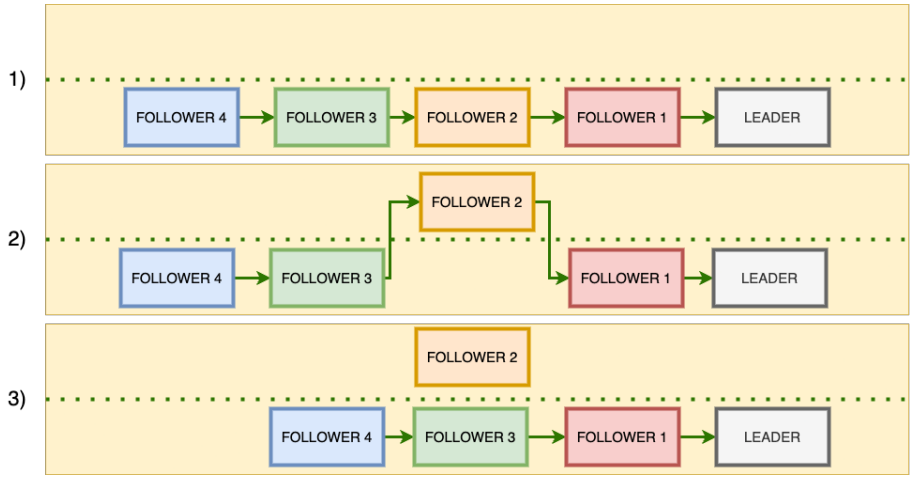


Fig. 8: Three-step evolution of the platoon when a vehicle leaves the formation

into state `JOINED`. Note that the enabled transitions are not evaluated together but according to the probabilistic approach of SMC, each simulation will have a different order of vehicles joining the platoon.

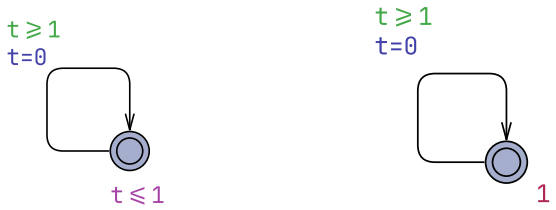
We introduce another location named `LEFT` to model a vehicle that left the platoon. For simplicity, we can assume that the leaving vehicle moves away from the longitudinal axis of the platoon, making space for the following vehicle. To model this behaviour, we imagine the leaving vehicle as a *ghost* vehicle that coincides with the front one. By doing so, the following vehicle can close up the distance with its new front vehicle. Figure 8 visualize the operation of a vehicle (namely, `FOLLOWER 2`) leaving the platoon. We obtain this behavior by changing the evaluation function in the location `LEFT` invariant, with a new *`ghost()`* function.

```
double ghost() {
    return c*a_leader+(1-c)*a_front - K1*(speed-v_lead) -
    K2*(position-x_front);
}
```

The new function differs just for the last term of the CACC Equation (4); indeed, instead of maintaining a distance `dsafe` from the preceding car, the ghost car will have the preceding car position itself as a reference, hence making it possible for the following car to get close.

4.4 Simulation control

Since invariants are constantly evaluated, to control the granularity of the enabled transitions we can introduce an automaton that enables a transition with a given frequency. Figure 9a shows an example automaton that has a single initial location, an invariant `t<=1`, a guard `t>=1`, and a reset `t=0`. This results in a



(a) Constant exact transition time of $1s$

(b) Exponential distribution of transition times with $\lambda = 1$

Fig. 9: Simulation automaton to control enabling transitions

transition exactly every 1 second. If we remove the invariant `t<=1` the transition is enabled every time `t>=1` but not exactly at 1 as in Figure 9b. The actual time will follow a certain probability distribution. We can also specify the rate of the exponential time distribution for the transitions, which has the following density function:

$$\lambda \cdot e^{-\lambda \cdot t}, \quad t \geq 0. \tag{5}$$

5 Simulation and probabilistic verification

In this Section, we proceed to first simulate the platoon system under different conditions to visually evaluate and validate the model. We will refer to a platoon configuration with 4 vehicles, where the first one is the leader. Then we apply SMC to provide probabilistic guarantees on the safety properties of the system. The overall system can be configured with several parameters, to explore different initial scenarios and requirements:

- Initial position and speed
- Desired safety distance between vehicles
- Control gain parameters
- 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 $100m$ and initial speeds so that the vehicles start still with null speed. An example of a simulation event trace can be seen in Figure 11. The figures show the event traces for two platoon configurations. In the first trace `Followers` join the platoon dynamically starting from `t>=10`. In the second trace, `Follower2` leave the platoon starting from `t>=30`. From the two runs, it is possible to note the probabilistic approach of UPPAAL to select the enabled transition to evaluate; indeed, the order in which `Followers` join is different every run, depending on the seed set by the simulation.

The control Equation (4) was set up with $c1 = 0.1,\ k1 = 1,\ k2 = 1,\ d_{dsafe} = 50m$. Figures 12 and 13 show two simulation outcomes, with a set desired distance of $50m$. Figures 12 show the nominal behaviour without unsafe states and

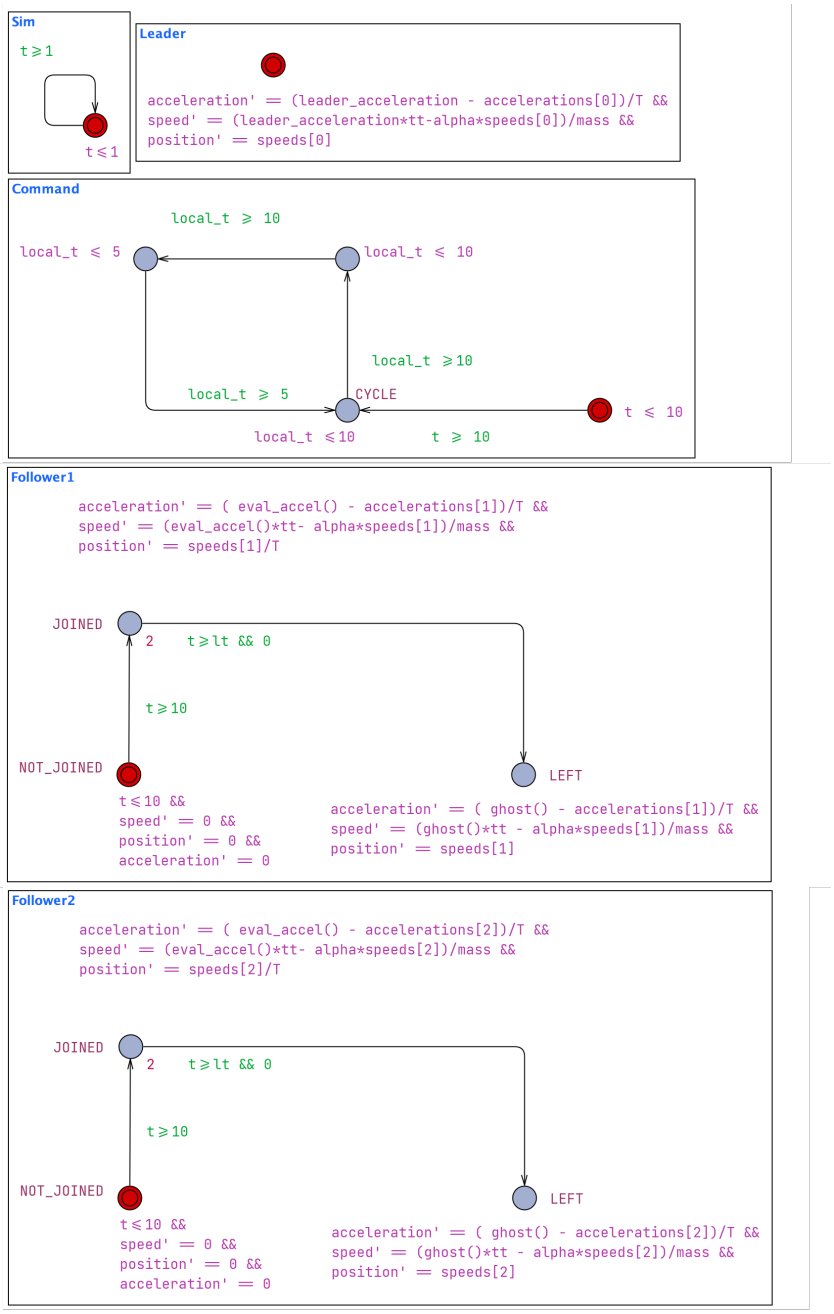


Fig. 10: Network of timed automata for the system with 2 followers and 1 leader.

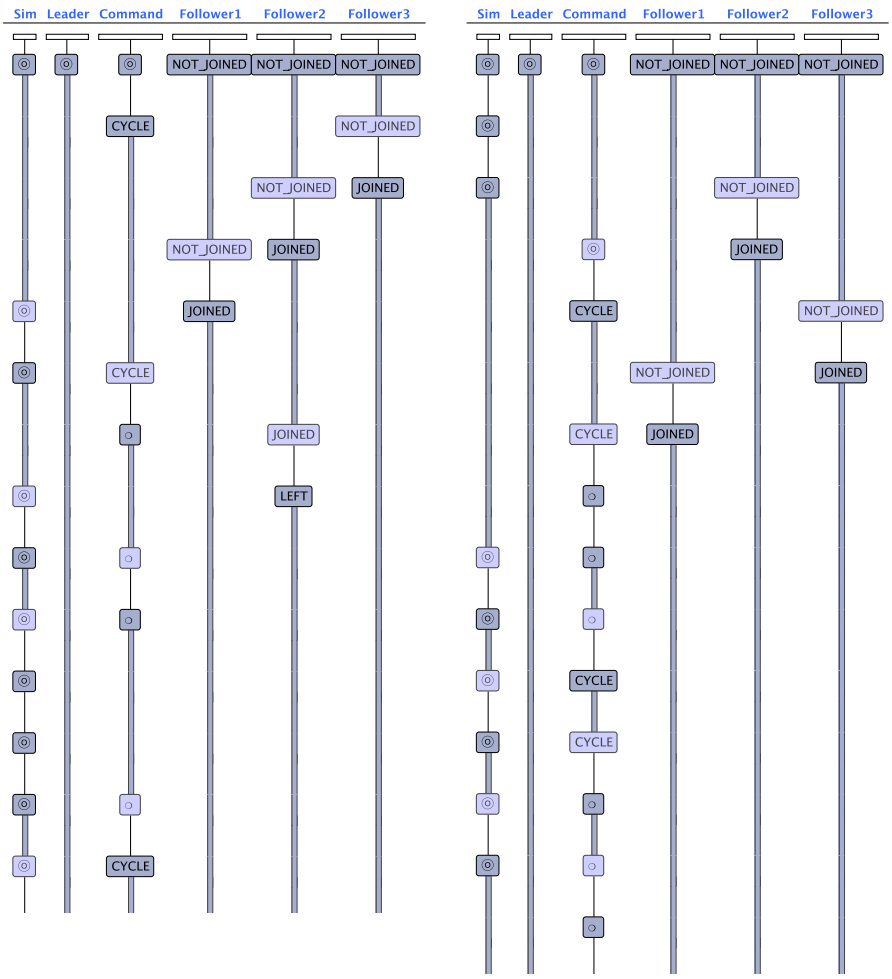


Fig. 11: Event trace with the transitions of vehicle joining the platoon and `Follower2` leaving it.

leaving vehicles. Figure 13 shows the behaviour of a vehicle leaving the platoon, simulated as a ghost reference after `t>=30`.

Instead, Figure 14 shows the simulation outcome with a different value of k1, k2 so that $c1 = 0.1,\ k1 = 0.1,\ k2 = 0.1,\ d_{dsafe} = 50m$. From the figure, it is clear that this configuration leads to an unsafe state due to the poor reactivity of the control system; indeed the parameters $k1, k2$ control the adjustments in the CACC equation. The higher $k1$ the higher the acceleration adjustment due to the speed error. The higher $k2$ the higher the adjustment to the position error.

$$k_1 \cdot (\dot{x}_i - \dot{x}_{leader}) \tag{6}$$

$$k_2 \cdot (x_i - x_{i-1} + d_{safe}). \tag{7}$$

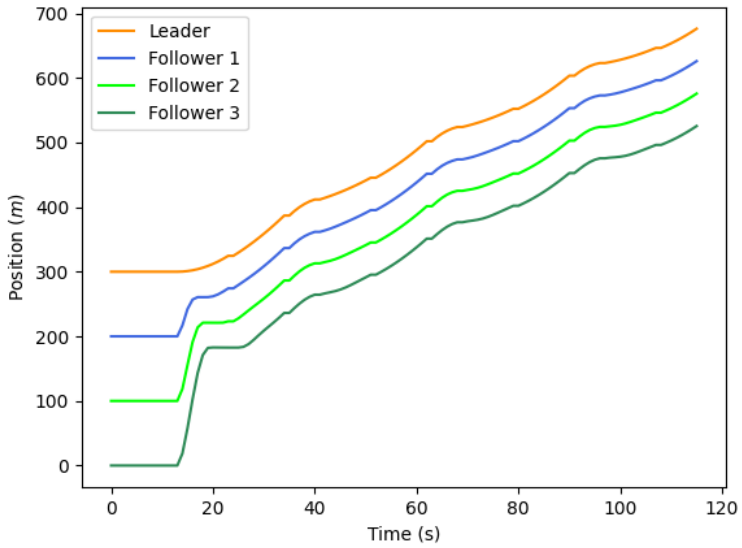


Fig. 12: Position of the 4 platoon vehicles with a desired distance of $50m$.

In Figure 15 we present another configuration with different safety distance requirements that show the simulation outcome with a $c1 = 0.1,\ k1 = 1,\ k2 = 1,\ d_{dsafe} = 10m$. From the figure, it is evident that also this configuration leads to an unsafe state due to the reduced safety distance between vehicles.

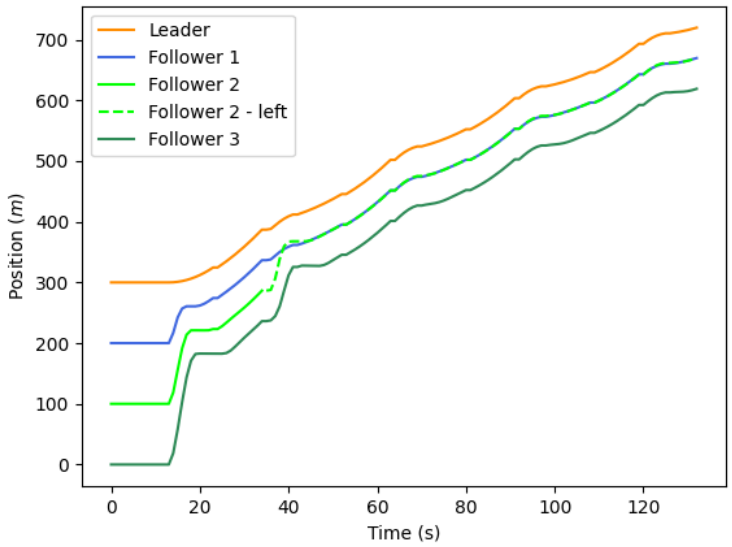


Fig. 13: Position of the 4 platoon vehicles with `Follower2` leaving at time `t>=30`, becoming a ghost reference.

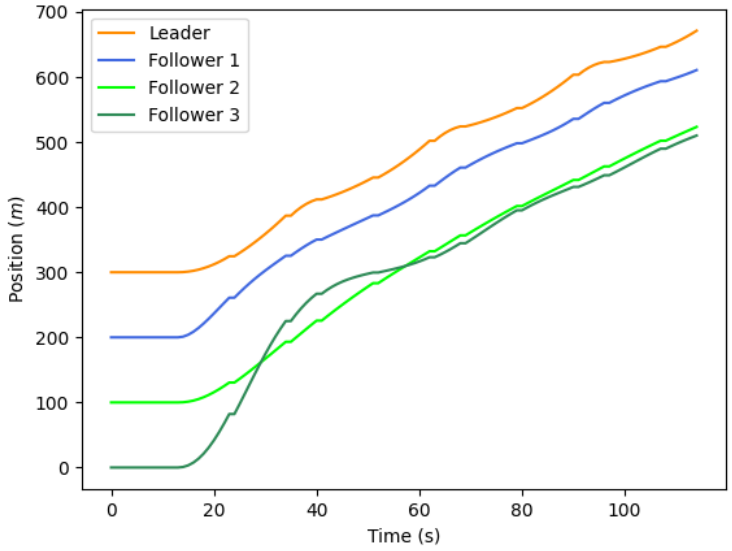


Fig. 14: Position of the 4 platoon vehicles with a crash due to slow reactivity from the control

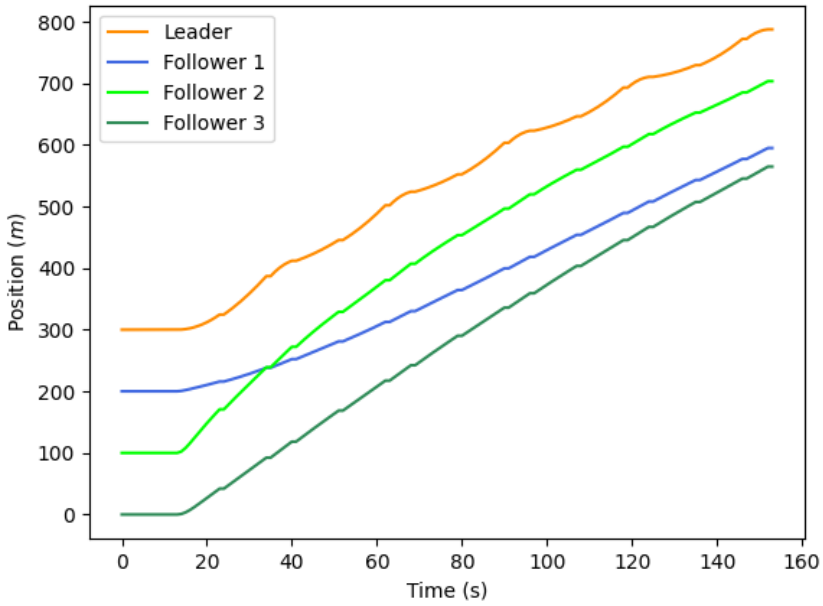


Fig. 15: Position of the 4 platoon vehicles with a crash due to small safety distance between vehicles.

Table 1: Safety properties on the position of the vehicles in the platoon

Purpose	**Query**
S1 `Follower1` position check	`Pr[<=200] ([] positions[0] > positions[1])`
S2 `Follower2` position check	`Pr[<=200] ([] Follower2.left || positions[1] > positions[2])`
S3 `Follower3` position check	`Pr[<=200] ([] positions[2] > positions[3])`

Table 2: Functional properties on the participation to the platoon

Purpose	Query
F1 `Follower2` leaving	`Pr[<=200] (<> !Follower2.leaving || Follower2.left)`
F2 `Follower1` joining	`Pr[<=200] (<>Follower1.joined)`
F3 `Follower2` joining	`Pr[<=200] (<>Follower2.joined)`
F4 `Follower3` joining	`Pr[<=200] (<>Follower3.joined)`

5.1 Functional and safety properties

In this Section, 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, `[]` indicates that `expr` must always hold. Moreover, a property that must eventually be held is defined as `Pr[<=simulation_time] (<> expr)`. 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`). We used the following parameters for the verification:

- Desired safety distance: $50m$
- $k1 = 1,\ k2 = 2,\ c1 = 0.1$
- Initial speeds for all vehicle: $0\frac{m}{s}$
- Initial positions: vehicles spaced by $100m$.

Table 3: Functional properties on maintaining the correct fixed distance between the vehicles in the platoon

Purpose	Query
F4 `Follower1` distance	`Pr[<=200] (<> t >= 100 && (Sim.distances[0] > Follower1.dsafe*0.95 && Sim.distances[0] < Follower1.dsafe*1.05))`
F5 `Follower2` distance	`Pr[<=200] (<> t >= 100 && (Sim.distances[1] > Follower2.dsafe*0.95 && Sim.distances[1] < Follower2.dsafe*1.05))`
F6 `Follower3` distance	`Pr[<=200] (<> t >= 100 && (Sim.distances[2] > Follower3.dsafe*0.95 && Sim.distances[2] < Follower3.dsafe*1.05))`

Table 1 summarises the safety properties we want to prove using statistical model checking. In detail, we want to assess the probability of the vehicle positions not overlapping during the platoon lifetime in the simulation. Furthermore,

Table 2 lists the functional properties for the platoon formation and Table 3 details the properties for the convergence of the platoon positions 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, with a safety distance of $50m$, we want to prove that, eventually, the average distance between the first and the second vehicle in the platoon will be within 5% of the safety distance, i.e. $Sim.distances[0] \in [47.5, 52.5]$.

Table 4 shows the statistical model checking verification results for the properties above. In detail, we report the probability bounds of the property being true, the confidence interval of this bound and the number of runs needed by the statistical model checker to obtain such probability bound and confidence interval. As we can see, we managed to prove the safety properties with a probability of at least 99% with a confidence interval of 95%.

Table 4: Statistical model checking proof of the properties

Property	C.I.	Probability	Runs
S1	95%	$[0.99, 1)$	368
S2	95%	$[0.99, 1)$	368
S3	95%	$[0.99, 1)$	368
F1	95%	$[0.95, 1)$	72
F2	95%	$[0.95, 1)$	72
F3	95%	$[0.95, 1)$	72
F4	95%	$[0.99, 1)$	368
F5	95%	$[0.99, 1)$	368
F6	95%	$[0.99, 1)$	368

6 Conclusion

This work has shown how statistical model checking can be used for the analysis of the behaviour of cooperative cyber-physical systems, where physical processes and digital control are strictly interconnected. In particular, SMC is shown to be a strong framework that can evaluate system characteristics in a probabilistic manner in a variety of operational scenarios. With respect to studies based on simulation [26], not only can the incorporation of SMC improve the comprehension of system behaviour, but it also promotes well-informed choices during the design and implementation phases of the system. Further work will consider the application of SMC to prove safety under complex physical aspects of the vehicle dynamic, critical road surfaces (e.g., wet or icy), and more realistic cooperative autonomous driving systems.

References

1. S. Sheikholeslam and C. A. Desoer, "Longitudinal control of a platoon of vehicles," in *1990 American Control Conference*, pp. 291–296, 1990.
2. X.-Y. Lu, J. Hedrick, and M. Drew, "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.
3. S. Maiti, S. Winter, and L. Kulik, "A conceptualization of vehicle platoons and platoon operations," *Transportation Research Part C: Emerging Technologies*, vol. 80, pp. 1–19, 2017.
4. K.-Y. Liang, J. Mårtensson, and K. H. Johansson, "Heavy-duty vehicle platoon formation for fuel efficiency," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1051–1061, 2016.
5. S. E. Li, Y. Zheng, K. Li, and J. Wang, "An overview of vehicular platoon control under the four-component framework," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 286–291, IEEE, 2015.
6. E. M. Clarke, "Model checking," in *Foundations of Software Technology and Theoretical Computer Science* (S. Ramesh and G. Sivakumar, eds.), (Berlin, Heidelberg), p. 54–56, Springer, 1997.
7. R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.
8. E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Logics of Programs* (D. Kozen, ed.), (Berlin, Heidelberg), p. 52–71, Springer, 1982.
9. E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst.*, vol. 8, p. 244–263, apr 1986.
10. E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*, p. 1–30. Berlin, Heidelberg: Springer, 2012.
11. A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *Runtime Verification* (H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, eds.), (Berlin, Heidelberg), p. 122–135, Springer, 2010.
12. Z. Huang, D. Chu, C. Wu, and Y. He, "Path planning and cooperative control for automated vehicle platoon using hybrid automata," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 3, pp. 959–974, 2019.
13. M. Barbier, A. Renzaglia, J. Quilbeuf, L. Rummelhard, A. Paigwar, C. Laugier, A. Legay, J. Ibañez-Guzmán, and O. Simonin, "Validation of perception and decision-making systems for autonomous driving via statistical model checking," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, p. 252–259, June 2019.
14. A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, p. 397–415, Aug. 2015.
15. "Formal verification of autonomous vehicle platooning," vol. 148, p. 88–106, Nov. 2017.
16. A. Desai, A. Desai, T. Dreossi, T. Dreossi, S. A. Seshia, and S. A. Seshia, "Combining model checking and runtime verification for safe robotics," p. 172–189, Sept. 2017. DOI: 10.1007/978-3-319-67531-2-11 MAG ID: 2752666516.
17. B. Barbot, B. Bérard, Y. Duplouy, and S. Haddad, "Statistical model-checking for autonomous vehicle safety validation," in *Conference SIA Simulation Numérique*,

(Montigny-le-Bretonneux, France), Société des Ingénieurs de l'Automobile, Mar. 2017.
18. A. Paigwar, E. Baranov, A. Renzaglia, C. Laugier, and A. Legay, "Probabilistic collision risk estimation for autonomous driving: Validation via statistical model checking," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, p. 737–743, Oct. 2020.
19. M. Novak, U. M. Nyman, T. Dragicevic, and F. Blaabjerg, "Statistical model checking for finite-set model predictive control converters: A tutorial on modeling and performance verification," *IEEE Industrial Electronics Magazine*, vol. 13, p. 6–15, Sept. 2019.
20. P. Filipovikj, N. Mahmud, R. Marinescu, C. Seceleanu, O. Ljungkrantz, and H. Lönn, "Simulink to uppaal statistical model checker: Analyzing automotive industrial systems," in *FM 2016: Formal Methods* (J. Fitzgerald, C. Heitmeyer, S. Gnesi, and A. Philippou, eds.), (Cham), p. 748–756, Springer International Publishing, 2016.
21. G. Behrmann, A. David, and K. G. Larsen, *A Tutorial on Uppaal*, p. 200–236. Berlin, Heidelberg: Springer, 2004.
22. J.-P. Katoen, "The probabilistic model checking landscape," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, (New York, NY, USA), p. 31–45, Association for Computing Machinery, 2016.
23. G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "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.
24. T. A. Henzinger, "The theory of hybrid automata," in *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pp. 278–292, IEEE, 1996.
25. D. Swaroop and J. Hedrick, "String stability of interconnected systems," in *Proceedings of 1995 American Control Conference - ACC'95*, vol. 3, pp. 1806–1810 vol.3, 1995.
26. M. Palmieri, C. Quadri, A. Fagiolini, and C. Bernardeschi, "Co-simulated digital twin on the network edge: A vehicle platoon," *Computer Communications*, p. 35–47, 2024.