Statistical Model-Checking for Autonomous Vehicle Safety Validation

Benoît Barbot[4], Béatrice Bérard[2,3], Yann Duplouy[1,2], Serge Haddad[2]

1: IRT SystemX, Paris-Saclay, France
2: LSV, ENS Paris-Saclay, CNRS, Inria, France
3: Sorbonne Universités, UPMC Univ. Paris 06, CNRS UMR 7606, LIP6, Paris, France
4: LACL, Université Paris Est Créteil

Abstract: We present an application of *statistical model-checking* to the verification of an autonomous vehicle controller. Our goal is to check safety properties in various traffic situations. More specifically, we focus on a traffic jam situation.

The controller is specified by a C++ program. Using sensors, it registers positions and velocities of nearby vehicles and modifies the position and velocity of the controlled vehicle to avoid collisions. We model the environment using a stochastic high level Petri net, where random behaviors of other vehicles can be described.

We use HASL, a quantitative variant of linear temporal logic, to express the desired properties. A large family of performance indicators can be specified in HASL and we target in particular the expectation of travelled distance or the collision probability.

We evaluate the properties of this model using COSMOS[1]. This simulation tool implements numerous statistical techniques such as sequential hypothesis testing and most confidence range computation methods. Its efficiency allowed us to conduct several experiments with success.

Keywords: Statistical Model Checking, Simulation, Autonomous Vehicle, Controller

1. Introduction

Context. Intelligent Transport Systems (ITS) address the numerous challenges faced by the design of safe softwares dedicated to (almost) autonomous transportation systems.

Recently several ITS projects aim at providing assistance to drivers and focus on partially automatized roads. Researchers have first chosen an approach based on a fully automatized infrastructure (see project PATH [9]) in the nineties. This approach has been gradually replaced by a new research direction, more oriented on strategies to ensure specific properties like collision avoidance or preserving safety distances [10]. In particular the control of an autonomous vehicle may be switched between the driver and an automatic controller depending on the traffic situation and the road structure.

Goal. This work takes place in the SVA[2] project, hosted by IRT SystemX. Our long term aim is to be able to verify the controller developped in this project for a traffic jam situation. As a first step in this direction, our goal here is to propose a methodology to check whether in a given situation the controller ensures a specific property. Such a work raises difficult issues: (1) the modelling of the environment and the vehicle requires a careful study to define which parameters are relevant and (2) the variety of possible incidents that may occur during the travel triggers a combinatory explosion that may forbid any verification procedure.

Our approach. From a modelling point of view, we selected a stochastic high-level Petri net based approach. Indeed (1) the parallelism induced by the moves of the cars requires a model able to handle this feature (hence Petri nets), (2) this model should manage user-defined data structures and code (hence high-level Petri nets) and (3) the model should quantify the frequency of the different incidents as well as the unpredictability of the behaviour of the other cars (hence stochastic high-level Petri nets).

In order to evaluate performance indicators of stochastic systems there are currently two main approaches. Numerical methods require to restrict the probability distributions occurring in the system in order to exhibit a Markovian-like behaviour. In addition, except in very particular cases, they cannot handle large scale systems. Statistical methods are more robust although they only provide confidence intervals for the performance indices. However for most of the applications, these intervals are sufficient.

Contributions. With the approach described above, we propose a stochastic high-level Petri net modeling a traffic jam situation. Besides the probabilistic features representing the environment, this model takes into account the control of the autonomous vehicle by a C++ program. It is then given as input to the tool COSMOS, with two performance indicators: the expected travelled distance and the probability of collision. Using simulations in various situations, we produce estimations of these indices with confidence intervals.

[1] Statistical Concepts and Tools for Stochastic Models

[2] Simulation pour la Sécurité du Véhicule Autonome (Simulation for the Safety of Autonomous Vehicle)

We have considered two simple controllers: a basic one that adjusts its speed on the vehicle preceeding it without changing its lane and a slightly more evolved one that may change its lane depending on the situation. We have taken into account two behaviours for the other vehicles: either smooth speed changes or unpredictible ones. Our experimentations have established that a small increase in the controller capacities triggers a large increase about the vehicle safety performances. On the other hand the simulation time that we observed confirm that our approach is able to manage large scale models.

Outline. The modeling formalism is described in Section 2 and the case study is presented in Section 3. We briefly explain the main features of the tool COSMOS in Section 4 and show our experimental results in Section 5. Finally, we present work in progress and perspectives in Section 6.

2. Formalism

In this section, we briefly describe the formalisms used in this work: Petri net variants to represent the model of the motorway and HASL formulas for the properties.

2.1 Petri nets

A **Petri net** is a bipartite graph, whose nodes are either *places* or *transitions*. Places may contain *tokens*, and depict the states of the system. Transitions represent the activities of the system, which correspond to state changes. Arcs from the input places of a transition are labeled by a condition on the tokens under which the transition can be fired. When the transition is fired (atomically), these tokens are consumed, while new tokens are produced in the output places of a transition, again according to the labels of the associated arcs. A *marking* is a mapping giving the number of tokens in each place.

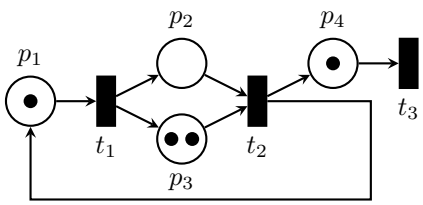

Figure 1: A Petri net

For instance, in the Petri net of Figure 1, the absence of labels on arcs indicates a default value of 1. From the initial marking, which can be represented by the vector $(1, 0, 2, 1)$, transition t_2 cannot fire because there is no token in input place p_2, and firing t_3 results in marking $(1, 0, 2, 0)$. On the other hand, from the same initial marking, sequence $t_1 t_2$ reaches marking $(1, 0, 2, 2)$.

High-level Petri nets are an extension of Petri nets where tokens carry information. Data types, called *color sets*, are associated with places and transitions and tokens respect the typing of places. The type of a transition specifies how it might be fired and is called a *binding* (since it binds local variables of the transition to values, see below).

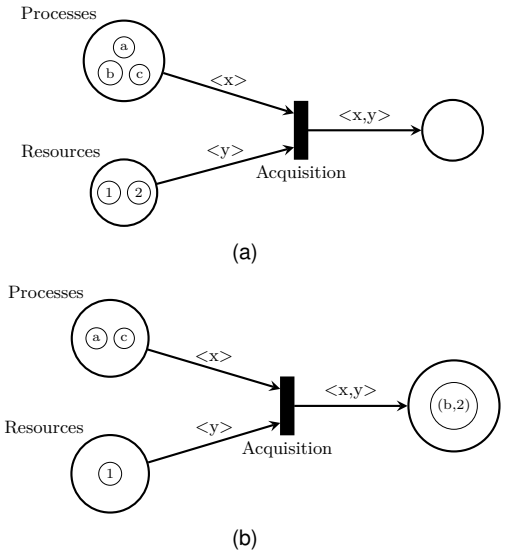

Figure 2: A high-level Petri net, before and after firing a transition

For instance, the high-level Petri net of Figure 2 describes how processes might acquire resources. On edges, the tokens are described by variables x (a process) and y (a resource). In this example, process b has acquired resource 2 which is represented by the presence of token $(b, 2))$ in the rightmost place. It might have been any other combination of process in $\{a, b, c\}$ and resource in $\{1, 2\}$.

To further increase the expressiveness of such Petri nets, we associate C code with some transitions, which will be executed at the end of the firing process. The global state of the net is thus defined as a tuple containing the marking (tokens present in the various places of the net) and the current values of the global variables of the C code. For instance one could manage a variable `count` that records the number of synchronisations between process and resource and would be incremented by the firing of the transition.

Stochastic high-level Petri nets are a further extension, where a probabilistic distribution is associated with each color of a transition, to specify the random delay between the enabling of the transition and its possible firing. These distributions can be marking-dependent and describe the **stochastic** behaviour of the model.

Thus the configuration of such a net consists of: (1) the marking of the net, (2) the values of global C variables, (3) for any transition binding associated with an enabled transition firing a scheduled time and (4) the current simulation time. The configuration change involves the following stages: (1) determine the next

firing among those with the earliest schedule, (2) advance the simution time, (3) update the marking w.r.t. the transition firing, (4) delete the disabled transition firings, and (5) for any newly enabled transition firings sample its delay distribution in order to get its schedule. The selection of the next transition firing is based first on the *priority* of the transitions and in case of multiple transition firings with same priority by a random choice related to the transition *weights*.

The model of the motorway depicted in Figure 4 is an example of such stochastic high-level Petri net, that will be explained in more details in Section 3.

2.2 HASL formulas

Properties of such stochastic high-level Petri nets are specified by **performance indices**, expressed in the logic HASL[3]. A formula of this logic is described by two components:

- a *linear hybrid automaton* which is, during execution, synchronized with the Petri net, in order to gather additional information, based on the LHA variables;
- an *expression* which includes the state variables of the automaton, path operators (i.e. $\max, \min, last$), alongside with probability operators such as the expectation over successful runs, or the probability of a successful run.

Example. We end this section with a toy example illustrating our approach on the computation of an estimation of π by the Monte-Carlo method. The idea is to select randomly points (x, y) in the square $[0, 1] \times [0, 1]$ and to check if they belong to the quarter disc with measure $\pi/4$ by testing $x^2 + y^2 \leq 1$. Figure 3 depicts a simple Petri net with two transitions and a HASL formula composed of a LHA and the expression $E(last(r))$. A trajectory is obtained by synchronizing the LHA with the Petri net, first on transition e_1, then on e_2, producing random values for x (firing delay of e_1) and y (firing delay of e_2) respectively. The last value of r is then 4 if $x^2 + y^2 \leq 1$, leading to one final state and 0 otherwise, leading to the other final state.

3. Case study

The case study concerns a (one way) motorway section shared by the controlled vehicle and other vehicles with trajectories that are randomized, but respect some constraints to keep them realistic. The controller is given by a C++ program and its goal is to avoid collision for the autonomous vehicle.

3.1 Model of the motorway

As shown in Figure 4, with the controlled vehicle drawn in red, the motorway section is discretized into cells, the cell with coordinates (i, j) representing position i on lane j.

The model of the motorway with vehicles is the stochastic high-level Petri net depicted in Figure 5, with parameters the length N_{pos} of the section, the number N_l of lanes, the maximal speed $v_{\max}$, and the maximal number N_{veh} of vehicles, given by the types of the tokens.

The execution of the model starts with transition `initGeneration`, which is fired only once and produces the initial configuration (via the dashed arcs). The orange part handles the behaviour of the controlled vehicle while the blue part handles the other vehicles, represented by tokens in the associated places:

- An environment vehicle is a token of the form $(x, y, \dot{x}, \dot{y})$ where (x, y) corresponds to the current cell and $(\dot{x}, \dot{y})$ is the speed vector, composed of horizontal and vertical speed (on each axis, in cells per time unit). These tokens can be either in place `otherVehicles` or in place `inProcess` when an update is performed.
- The controlled vehicle is a token stored in place `selfVehicle`. In addition to the elements used for

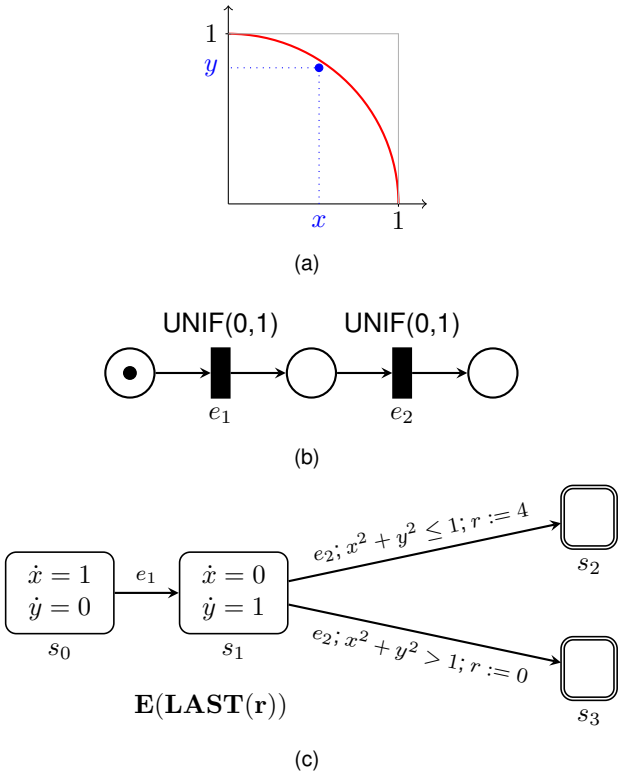

Figure 3: Computing an approximation of π (a) with a Petri net (b) and an associated HASL formula (c)

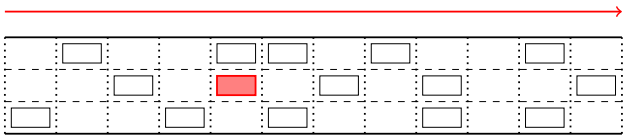

Figure 4: A motorway section

[3]Hybrid Automata Stochastic Language

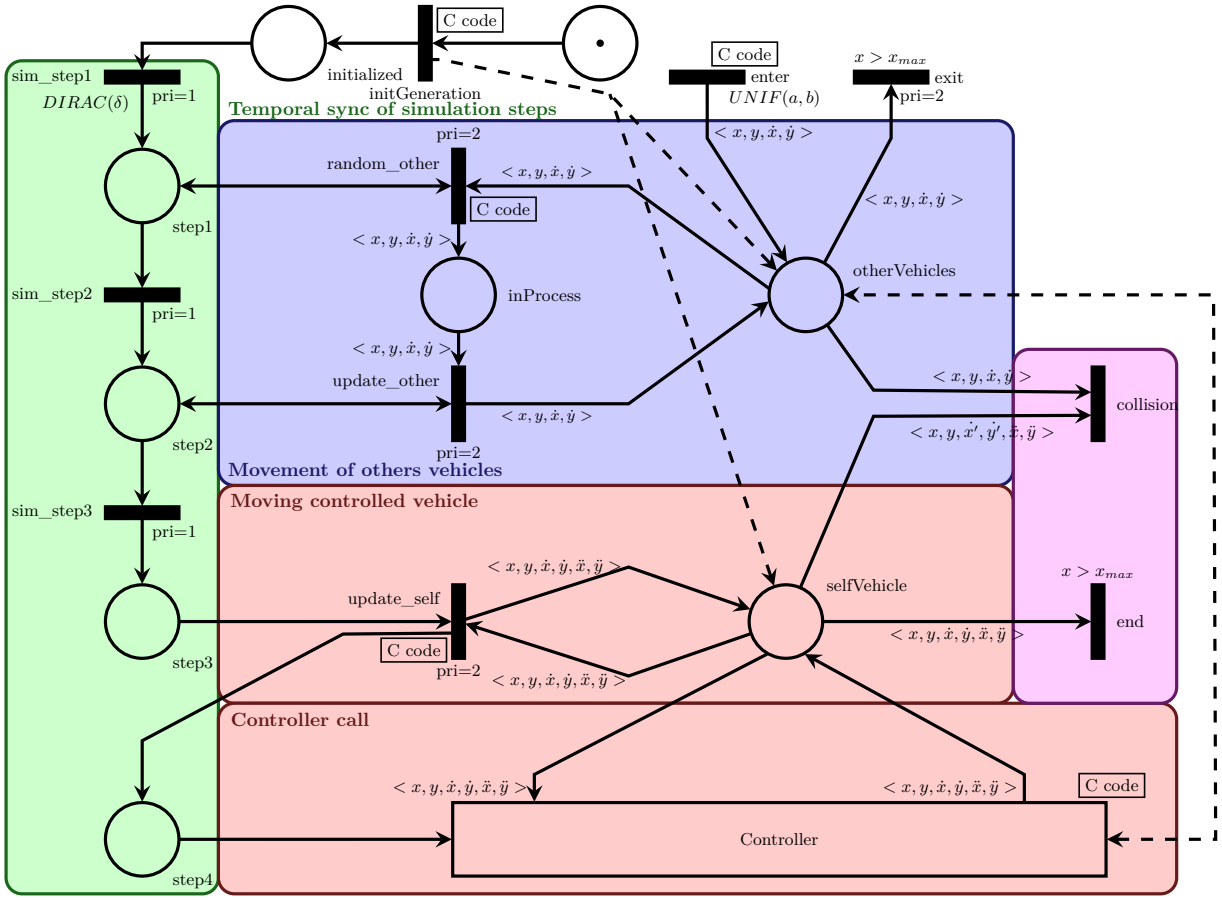

Figure 5: Motorway model

other vehicles, its description also includes the acceleration.

The green part of the Petri net controls the different phases of the simulation. The model runs quasi-synchronously, with a time step δ equal to the time step of the controller. Synchronously, the `sim_step1` transition fires for each δ (with the Dirac distribution) and starts the following process:

- In transition `sim_step2`, the new position of environment vehicles is computed using their current speed. The new speed is then randomly generated using a distribution that depends on environment parameters.
- In transition `sim_step3`, the new position and speed of controlled vehicle is computed, using its current speed and acceleration.
- The last step uses transition `Controller` which executes the associated C++ program, resulting in the update of the controlled vehicle acceleration.

Asynchronously, new vehicles are added at the start of the section, using transition `enter`. This transition is equipped with a uniform distribution on interval $[a, b]$ where a and b are parameters, which influence the load of the motorway.

The pink part on the right of the model handles the end of the simulation: either the controlled vehicle reaches the end of the section, which corresponds to the test $x > x_{\max}$, with $x_{\max} = N_{pos}$ or a collision happens between the controlled vehicle and one of the environment vehicles.

All informations about the controlled vehicle are given to the controller, together with a view on the other vehicles modeling the input received from sensors. For instance, Figure 6 shows a possible realization of this communication.

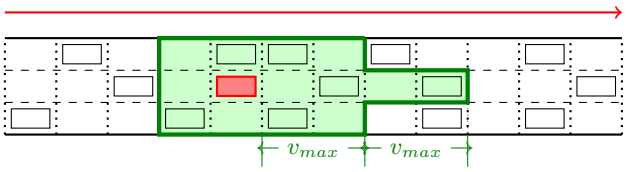

Figure 6: A possible view of the controller

3.2 Specification of the performance indices

We consider two performance indices: the probability of collision and the expected distance covered by the controlled vehicle before collision, if it occurs, or until the end of the section otherwise.

For the first one, we use the HASL formula represented in the upper part of Figure 7, where T denotes the set of all transitions of the Petri net in Figure 5. The synchronization implies that the state s_{col} is reached exactly when transition `collision` is fired in the simulation and the expression $\mathbf{PROB}$ means that the value returned is the probability of reaching s_{col}.

The second performance indice is obtained via the HASL formula depicted in the bottom part of Figure 7. With respect to the previous automaton, we add a new transition reaching state s_{end} that will be synchronized with the transition end of the Petri net, and a variable x that will take the current value of the horizontal position when the transition is fired. The expression $\mathbf{E}(\mathbf{LAST}(\mathbf{x}))$ (expected value of the last value of x) yields the desired indice.

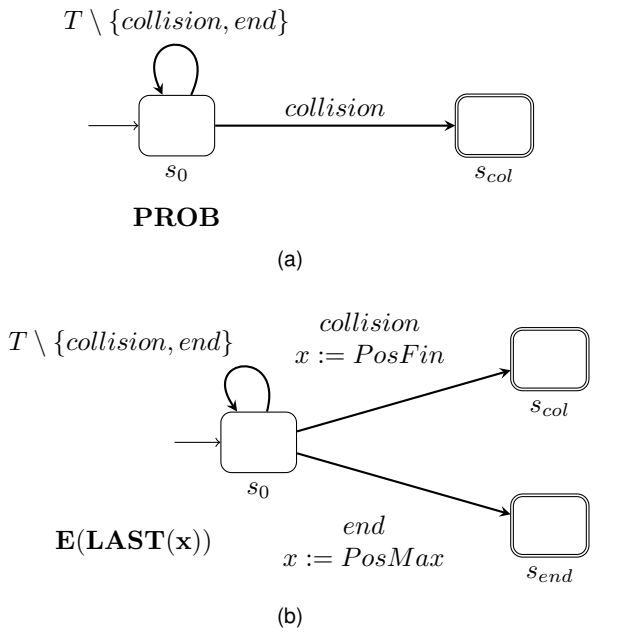

Figure 7: Two HASL formulas (a) for the probability of collision and (b) for expected travelled distance

4. Cosmos

We now give a brief description of the tool COSMOS used to compute the performance indices.

4.1 Architecture

COSMOS is a statistical model checker accepting Discrete Event Dynamic System (DEDS) as input models. In particular COSMOS takes as input several types of stochastic Petri nets with general distributions. As shown above in Figures 3 and 7, the specification to be checked is given as an HASL formula described by a Linear Hybrid Automaton (LHA) and an expression. The main algorithm of this tool randomly simulates the DEDS according to its stochastic semantics and synchronizes it with the execution of the automaton. During the synchronization, it evaluates HASL expressions. A statistical procedure decides when to stop the simulation and produces a confidence interval for a HASL expression.

The tool COSMOS consists of about 22000 lines of C++ code and is freely available at [2] under the Gnu General Public License version 3 (GPLv3). The tool relies on code generation to perform efficient simulation. It is divided into three main parts:

1. The parsing and code generation part reads the input files and the command line to build data structures for the DEDS and the automaton. Then optimised C++ code simulating both the behaviors of the DEDS and the automaton is generated. The resulting code is compiled by a C++ compiler and linked with the simulator library. The resulting binary is the simulator program.
2. The simulator part is a library implementing the algorithm synchronizing the DEDS and the automaton. It also implements the stochastic generation of events using the pseudo random number generator provided by the BOOST library and handles these events in an event queue.
3. The server part launches several copies of the simulator and aggregates their results. According to statistical parameters, a procedure decides whether enough trajectories have been simulated and stops all simulators when needed. Then, HASL expressions are evaluated and several output files are produced according to options. The computation of confidence intervals uses the BOOST library for the computation of quantiles of the normal distribution function and binomial distribution.

Several tools interact with COSMOS either by calling it or by being called by it:

- A C++ compiler is required to build the simulator from the generated code. Until now, GCC and Clang have been used.
- When the model is Markovian and the automaton has no clock, PRISM can be called by the simulator. In this case a state space generator is used and the CTMC of the product between the model and the automaton is given to PRISM. The vector of probabilities computed by PRISM on this model is parsed by COSMOS. This allows one to use PRISM to compute transient and steady state probabilities.
- The plotter GnuPlot can be called by COSMOS to produce a graph providing visualization of the results.
- Several OCaml scripts shipped with COSMOS perform benchmark, testing and plotting.

4.2 Application fields

In addition to the statistical model checking of DEDS on which COSMOS has shown its efficiency (see [1] for benchmarks with other tools), the particular architec-

ture of COSMOS have proven useful in several projects:

- The simulation engine has been specialised to handle simulation of rare events. By altering the sampling of distribution in the simulation algorithm efficient estimations of rare event probability have been computed [4].
- Thank to the very small footprint of the generated simulator, COSMOS has been used for the cosimulation of a pacemaker software with a model of the human heart. The generated simulator was small enough to fit in the memory of microcontroller on which live power consumption was measured [5].
- Custom probability distribution defined by polynomials have been used to sample uniform trajectory for timed automata [3].

4.3 Statistical procedures

We now detail the different statistical procedures proposed by COSMOS for evaluating HASL expression depending on several criteria.

- **Sequential hypothesis testing [11].** This procedure checks whether a probability is above a threshold. Parameters of this procedure are the probability of an error for a positive answer and a negative answer and the width of the indifference region. When the value of the probability is outside the indifference interval, the probability of an error is bounded by the parameter corresponding to the answer.
- **Chernoff-Hoeffding bounds [8].** This static method requires three related parameters, each of them can be determined by the two others. These parameters are the interval width, the confidence level and the number of samples. It outputs a confidence interval whose width satisfies the requirement and where the probabilistic guarantee is exact. It applies to estimate the expectation of a bounded random variable.
- **Chow-Robbins bounds [6].** This sequential method requires two parameters: the interval width and the confidence level. It outputs an interval whose width satisfies the requirement and where the probabilistic guarantee is asymptotic w.r.t. the width of the interval. It applies to estimate the expectation of a random variable, when no known bound is available.
- **Gaussian approximation.** This static method requires two parameters. The number of samples has to be given. The second parameter is either the confidence level or the interval width, one of these determining the other one. It ouputs an interval whose width satisfies the requirement and where the probabilistic guarantee is asymptotic w.r.t. the number of samples. It applies to estimate the expectation of a random variable. It is based on the central limit theorem.
- **Clopper-Pearson bound [7]** This static method computes confidence intervals for binomial distributions. It takes as input three parameters, the total number of samples, the confidence level and the number of successful samples and outputs a confidence interval for the probability of a sample to be successful.

5. Evaluation

Description. To evaluate our approach we have modelled simple controllers. We first study a basic controller which only aims at monitoring the vehicle in front of it and aligns its velocity to avoid collision with it. When there is no vehicle in front of it or if it is far ahead, this basic controller speeds up until it reaches a maximal velocity v_{max}. Then we study a more advanced controller which is able to change its lane to take over the vehicle in front of it. In order to change its lane the controller monitors vehicles in the two adjacent lanes. The controller initiates a take over only when the speed of the vehicle in front of him is slower than the speed v_{max} and when there is no vehicle in one of the adjacent lanes. In the positive case, the controller goes to the vehicle-free lane.

We have evaluated the controllers in the environment described in section 3 with two kinds of behaviours for other vehicles. One set of experiments in which all speed changes of other vehicles are smooth, at each step of simulation vehicles either stay at the same speed, moderately accelerate (the speed is increased by 1) or moderately decelerate (the speed is increased by 1). Those behaviours produce a highly predictive environment making the controller job easier. In contrast the second set of experiment involves vehicles which speed changes considerably: at each step of simulation, every vehicle randomly selects a new speed in a predefined interval.

Evaluation. The first objective of this benchmark is to compare the performances of the two controllers in different situations. As explained in Section 3.2, two performance indices have been measured: the probability of a collision and the expected covered distance before collision if any. The second objective is to quantify the influence of these parameters on the simulation time.

Observation. The results of our simulations are collected in Table 1. For these computations we simulate a motorway with 2 lanes and 1000 cells. The controlled vehicle is surrounded by 20 other vehicles whom speed varies between 3 and 5 cells per step. More precisely, all vehicles (including the controlled one) are generated randomly, with an uniform distribution on both lane and cells (up to the 200th). If a vehicle is generated on a cell where there is another vehicle, we generate new values for both lane and cell. We perform all the simulations on a single machine with 12 Intel Xeon cores using 8 parallel threads.

Comparing the probability of collision between the two

	Smooth velocity change				Aggressive velocity change			
	Basic		Lane-changing		Basic		Lane-changing	
Collision probability	0.838 ± 0.005		0.023 ± 0.005		0.847 ± 0.005		0.022 ± 0.005	
	468.51s	36 120	153.41s	6 620	459.67s	34 520	147.94s	6 020
Average distance before collision	267.44 ± 6.68		102.61 ± 2.56		282.23 ± 7.06		106.22 ± 2.66	
	71.48s	5 400	4 147 s	169 640	74s	5 480	3 766 s	154 120

Table 1: Evaluation of controllers. For each measure, controller and other vehicle behaviours we provide a 99% confidence interval of the measure, the time in second require to perform the simulation and the number of simulated trajectories.

controllers, we observe that the lane changing controller is far better than the basic one: 2.3% probability of collision vs 83.8% in the smooth environment and 2.2% vs 84.7% it the more aggressive one. We notice that the behaviours of other vehicles do not have a major impact on the probability of collisions. Similarly, the average travelled distance before collision is highly different between the two controllers.

We also have identified several limits of our simulation which we should overtake to produce more precise results.

Compared to real vehicles which feature turn signals and braking lights, there is no communication between vehicles in our simulation. This prevents the controller from synchronising its action with those of other vehicles. The behaviours of vehicles on the motorway should be enhanced to feature some communication.

We observe in table 1 that when the probability of collisions became small, the number of simulation required to obtain relevant estimates became large. This is clear when comparing the required time to estimate the average distance before collision for the two controllers (71s vs 4147s and 74s vs 3766s). Thus the simulation could be intractable when the probability of collisions becomes close to zero. This is a well-known difficulty in presence of rare event for which several theoretical techniques and practical tools have been developed in the literature and can be used to improve this simulation.

In Table 1 we observe that most collisions occurred at the beginning of the simulation. They occur around cell 104 for the lane changing controller and around cell 275 for the basic one. This shows that the simulation is biased by its initial state. More reliable results could be obtained by simulating a sliding window of the motorway around the controlled vehicle. Instead of simulating the controlled vehicle on the entire motorway we could make the controlled vehicle static in the middle of a short motorway and simulate all vehicles relatively to the controlled one.

6. Conclusion

In this work, we conducted with COSMOS several experiments on autonomous vehicle controllers, where the environment is modeled by a stochastic high level Petri net. With this model, it is possible to change easily the parameters of the simulations, producing different settings for the flow of other vehicles and their speed. However, the motorway section still corresponds to a basic situation.

Therefore, future research directions concern the design of refined models, where other situations could be taken into account. This could include for instance the addition of an emergency lane where the controlled vehicle could be led if there is no other way to avoid collision. Other interesting situations correspond to the presence of arrival or exit lanes, or of an alredy existing collision between other vehicles blocking one of the lanes.

Finally, considering a controller designed with Simulink would lead to check more elaborated versions of it.

Acknowledgement

This research work has been carried out in the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the French Programme "Investissements d'Avenir".

References

[1] P. Ballarini, B. Barbot, M. Duflot, S. Haddad, and N. Pekergin. Hasl: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90(0):53 – 77, 2015.

[2] B. Barbot, P. Ballarini, and H. Djafri. http://www.lsv.ens-cachan.fr/Software/cosmos/.

[3] B. Barbot, N. Basset, M. Beunardeau, and M. Kwiatkowska. Uniform Sampling for Timed Automata with Application to Language Inclusion Measurement. In *13th International Conference on Quantitative Evaluation of SysTems (QEST 2016)*, volume 9826 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2016.

[4] B. Barbot, S. Haddad, and C. Picaronny. Coupling and importance sampling for statistical model checking. In *Proceedings of (TACAS'12)*, pages 331–346, 2012.

[5] B. Barbot, M. Kwiatkowska, A. Mereacre, and N. Paoletti. Building Power Consumption Models from Executable Timed I/O Automata Speci-

fications. In *19th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2016)*, pages 195–204. ACM, 2016.

[6] Y. S. Chow and H. Robbins. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *The Annals of Mathematical Statistics*, pages 457–462, 1965.

[7] C. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, pages 404–413, 1934.

[8] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[9] R. Horowitz and P. Varaiya. Control design of an automated highway system. In *Proc. of the IEEE 88(7)*, 2000.

[10] Intelligent Vehicle Initiative. Saving lives through advanced vehicle safety technology, September 2005.

[11] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 06 1945.