Formal Modeling and Simulation of an Assisted Vehicle Overtaking

Abdelhakim Baouya[1] and Salim Chehida[1]

University Grenoble Alpes, France
{abdelhakim.baouya, salim.chehida@univ-grenoble-alpes.fr}

Abstract. Self-driving vehicles combine driver aids with extra software and sensors. They can achieve accurate automatic navigation, trajectory tracking, and automatic overtaking by using GPS, RADAR, and embedded cameras. Among them, overtaking is essential to avoid excessive waiting time and improve traffic efficiency. However, embedded sensors cause wrong driving state adoption for self-driving, because they cannot accurately locate nearby vehicles. This paper proposes formal modeling and simulation of a self-driving vehicle overtaking to establish a self-state adoption based on an auxiliary sensing system. The system architecture is described by using BIP language while checking LTL properties expressing functional and extra-functional requirements using statistical model checking tool.

Keywords: Automotive systems · Self-driving vehicles · Statistical model checking

1 Introduction

The self-driving vehicle is also known as an autonomous car or driverless car is a vehicle that has the ability for sensing its surrounding environment and moving without the human intervention [28]. It has attracted extensive attention in both academia and industry, looking towards safety and traveling efficiency [7,31]. According to World Health Organization [17], approximately 1.35 million people die each year as a result of road traffic crashes induced by humans (e.g., distracted driving, nonuse of seat-belts, and child restraints) or in-vehicle hardware factors (e.g., lousy quality electronic systems). Several large car manufacturers, thought-leaders, and innovators have the perspective that autonomous cars can substantially decrease traffic accidents and also alleviate pollution problems [21].

The architecture of the self-driving system is typically organized into two main parts [18]: the perception part and the decision making part. The perception system is responsible for estimating the state of the car using data captured by on-board sensors, such as Light Detection and Ranging (LIDAR), Radio Detection and Ranging (RADAR), camera, Global Positioning System (GPS), etc., while decision-making system determines the driving state for the next moment, such as acceleration/deceleration and overtaking.

Overtaking for self-driving vehicles is a complicated task compared to decision-making activities. While decision-making activities are based on offline learned models [12,26,15], the overtaking [19], involving two vehicles is established as a three phases maneuvers: Firstly, the self-driving vehicle changes the lane according to the planned trajectory.

Secondly, it drives along with the overtaken vehicle at a prescribed lateral distance. Finally, it will return to the lane and reach a preselected position in front of the overtaken vehicle.

Available literature [26,22] has planned their overtaking trajectories under the premise that the onboard sensors can detect the surrounding obstacles when overtaking. However, under exceptional circumstances, cameras and RADAR may be blocked by a bus or a truck. Therefore the configuration in Fig. 1 imposes the self-driving car to follow at a reduced velocity or necessitates the human intervention, and obviously, it incurs time-consuming and traffic congestion.

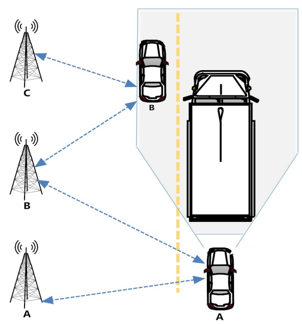

Fig. 1: The self-driving vehicle is blocked.

Solutions that improve nearby vehicle localization are relative to an auxiliary sensing system that assists the self-driving car during the overtaking when the camera and RADAR are inefficient. Due to the obstruction, the car sends a request to nearby servers to obtain the localization of the vehicle near the self-driving car. In this server, an information fusion algorithm with regard to the wireless signal and GPS is designed to estimate the position of the nearby vehicle [25]. The localization of the nearby vehicle will be transmitted to the self-driving car. Based on the received information, the self-driving car will determine the driving state in the next moment. In this paper, the driving state includes overtaking, lane changing, acceleration, and deceleration. The challenge in this area is the reuse of modeling and verification frameworks able to accommodate the operating complexity of the overtaking while allowing the design correctness regarding design requirements. One way of ensuring formally the requirement satisfiability related to overtaking is to employ statistical model checking. For sake of clarity, the contribution of the paper is split into three points:

- The overtaking system involving an auxiliary sensing system is captured in a component-based language called BIP (Behavior, Interaction, Priority).
- Simulation is carried out to validate the approach in case of undetected nearby vehicles.
- Statistical model checking is performed over the modeled system to check properties expressed in Probabilistic Bounded Linear-time Temporal Logic (PBLTL).

This paper is organized as follows: Section 2 identifies the actual work. Section 3 states the problem related to overtaking assistance. Section 4 portrays the architecture of the self-driving system. The analysis covering verification and simulation is portrayed in Section 5. To sum up, we draw our conclusions and also perspectives in the last section.

2 Related work

Up to now, there is significant research in the self-driving realm concerning the perception hardware system and the decision-making algorithms. Many companies like Google, Baidu, Uber, and Tesla demonstrated interest in developing self-driving cars, and collaborate with many hardware companies to build the essential electronic components. For instance, Baidu, one of the giant companies in china, developed an open-source self-driving car project called Apollo [1]. The published project portrays the inner modules related to the perception and decision making. Also, several companies are involved, such as Bosh, Intel, Ford, Nvidia, and Microsoft. The so-called *Tesla autopilot* is now able to match speed with traffic conditions, keep within a lane, and change lanes. In perception hardware, they are combining LIDAR and camera to estimate the localization of the self-driving car relative to the map. LIDAR can perceive the environment in the same way as RADAR due to shorter wavelength and high resolution. However, the size and the complexity of LIDAR hinder its commercialization, but academic researchers are making efforts to make LIDAR smaller and cheaper [13].

Further, concerning decision algorithms, deep learning algorithms have been used to detect pedestrians, vehicles, and other objects that could serve as obstacles on the vehicle's path [23,20]. For instance, the Udacity project for self-driving cars employs Neural Networks (NN) to annotate images and propose free open access to the dataset for educational and simulation purposes. It contains over 65,000 labels (e.g., car, truck, pedestrian) across 9,423 frames collected from a Point Grey research cameras running at a full resolution of 1920x1200 at 2hz [24]. According to the massive information collected and analyzed to understand the surrounding environment, self-driving vehicles can determine the next state such as overtaking, acceleration, or deceleration using learning algorithms [12,26,15]. However, learning necessitates constant progress in processing unit manufacturing [16], AI will struggle to advance without continuous breakthroughs in infrastructure capabilities.

Simulation is commonly used in engineering practice, and it has not been used to reason about formal specifications [3]. Statistical model checking (SMC) uses a simulation-based approach to reason about properties expressed in temporal logic. Using SMC, executions are first sampled, after which statistical techniques are applied to determine whether such a property holds. SMC techniques have been applied for the analysis of various case studies such as autonomous driving controllers [5] and biological systems [9]. Several frameworks exist for modeling and analysing stochastic systems, especially, statistical model checking has drawn lots of interest in the research community as UPPAAL-SMC [2], PRISM-SMC [10], MRMC [14], YMER [29], and COSMOS [8]. It is often used as an alternative to the time and memory intensive methods like model checkers. For instance, PRISM implements SMC techniques such as Probability Estimation techniques (PE) [11] and Hypothesis Testing (HT) [30], and the model to be checked is constructed before and stored in memory. MRMC offers SMC with confidence interval computation. However, it always loads Markov chain representations into memory completely. Ymer considers Generalized Semi Markov Processes (GSMP) and Continuous

Time Markov Chains (CTMC) using the PRISM dialect and uses a numeric-symbolic engine from PRISM. COSMOS uses confidence interval computation and exhibits performance comparable to PRISM on several benchmarks [4]. UPPAAL-SMC and Ymer are closer to BIP SMC, and both of them consider GSMP. UPPAAL-SMC provides a general stochastic timed semantics and is limited to exponential and uniform density functions. Furthermore, BIP is a component based language endowed with capabilities to express automata-based and/or Petri Net behavior. To perform SMC, the BIP engine relies on the constructed models in C++. For a deeper understanding of the SMC tools, we refer to the survey in [3].

3 Problem statement

The problem to be discussed in this section is portrayed in Fig.1, the self-driving vehicle, referred to vehicle A is blocked by the truck in front. The vehicle cannot change the lanes or detect vehicle B on the left side of the truck in spite is equipped with a camera and RADAR. The roadside servers are proposed to assist vehicle A to detect the vehicle out of the field of view. So, the self-driving vehicle can determine the driving state for the next moment. Subsequently, these nearby servers will estimate the vehicle's location near the self-driving vehicle using a designed information fusion algorithm [27] detailed in the appendix A. The self-driving vehicle will receive nearby vehicles' location information from these servers.

In the presence of obstruction, an accurate vehicle localization algorithm for self-driving vehicles based on the GPS localization. Assuming that the location for vehicle b is $b_{v,x}$ and $b_{v,y}$, the location of the server s_x and s_y, server selection should satisfy the formula:

$$min(\sqrt{(b_{v,x} - s_x)^2 + (b_{v,y} - s_y)^2}) \tag{1}$$

The servers within communication coverage of self-driving vehicles will send their information, including their ID and localization to the self-driving vehicle. Subsequently, the self-driving vehicle will select two nearest servers on the same side to assist accurate vehicle localization. As shown in Fig.2, the vehicle will select servers A and B since they are the two nearest servers when the self-driving vehicle is at position 1. After a period of time, when at position 2, servers B and C are selected.

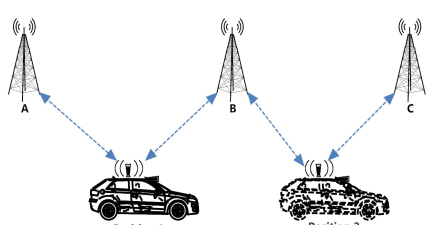

Fig. 2: Vehicle positions when passing through one server.

We want to validate our specifications by simulating vehicles travelling on parallel lanes. An operation is needed that implements the physical behaviour triggered every 0.5 seconds. Four functions constitute the bulk of the overtaking.

1. `isObstacleInFront()`: This function returns a boolean value if the truck or bus is detected based on the annotated captured images.
2. `findClosestServers()`: This function computes the closest servers to the vehicle.
3. `overtakingAcceleration()`: This function updates the current acceleration with the required one for overtaking.
4. `overtakingDeceleration()`: This function updates the current acceleration value with the required one for Deceleration.
5. `changeLane()`: This function changes the position of the vehicle on the left road lane.
6. `SafeVelocity()`: This function returns the adequate velocity when the computed one is refused.

The validation also requires four variables to obsever the updates related to the acclearation (a), the velocity (v), and the position (x) gouverned by the time ($delta$). The classical motion equations for position Eq.2 and velocity Eq.3 are implemented.

$$x_{next} = x_{past} + v_{past} * delta + 0.5 * a * delta * delta \tag{2}$$

$$v_{next} = v_{past} + a * delta \tag{3}$$

4 A component-based architecture

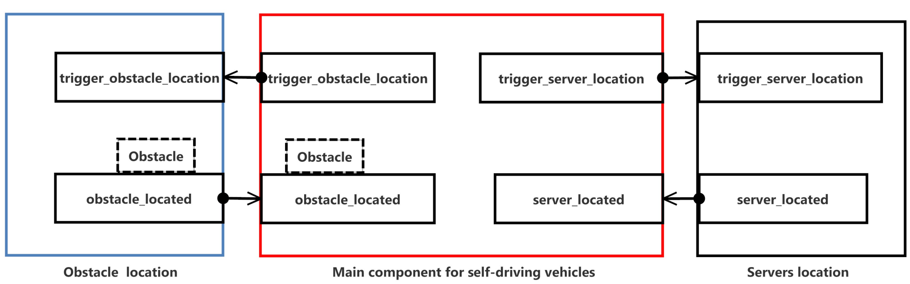

Fig. 3: Deployed architecture of the case study in BIP.

This work uses BIP, a highly expressive component-based language [6] that allows building systems in a hierarchy structure following a component-based philosophy starting from atomic components characterized by a behaviour expressed in automaton fashion and their interfaces (i.e. ports) to convey data to/from components. The language has a stochastic semantics and efficient tools for analysis based on statistical model checking techniques.

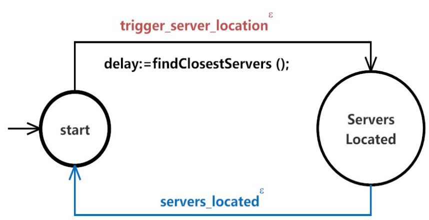

Fig. 4: Servers location automata.

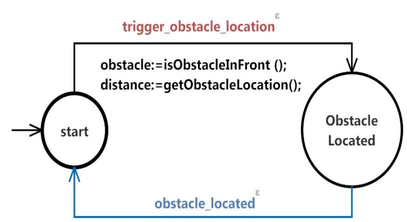

Fig. 5: Obstacles location automata.

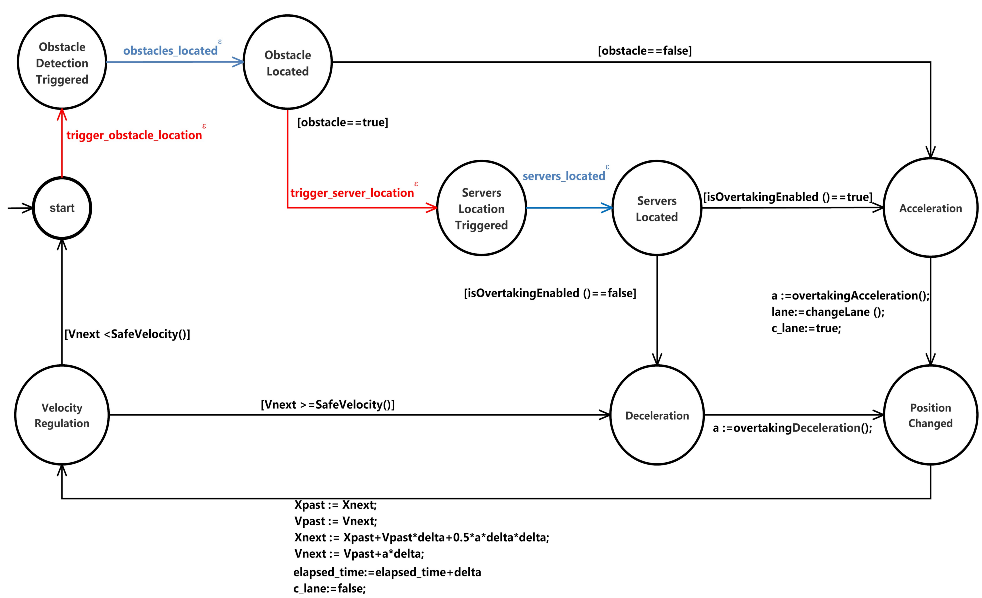

Fig. 6: Acceleration/Deceleration automata for self-driving vehicles.

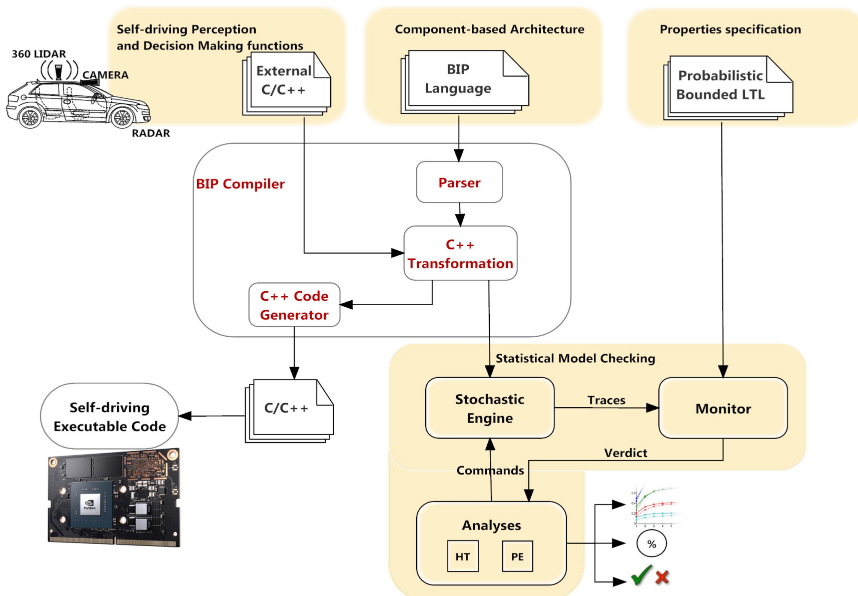

Fig. 7: BIP code generation and verification for self-driving vehicles.

The architectural assembly of our system as portrayed in Fig.3 is composed of three components: The component orchestrating the vehicle overtaking (middle), a component in charge of obstacle detection (left), and a component in charge of collecting the closest servers (right) where the behavior in the form of automata in Fig.6, Fig.5, and Fig.4, respectively. These three components are synchronized through *ports*. Ports are action names that can be associated with data and used for interactions with other components (i.e. Ports are illustrated in Fig.5 using "ε"). Also, ports could be silent (not synchronized) for labelling internal transitions as the one leaving *Obstacles_Located* state to *Acceleration* state in Fig.6. For instance, the port *trigger_server_location* in Listing.1.1 line 16 is used to synchronize the component server location automata and the main self-driving vehicle, and the port *servers_located* allows the main component to take back the control. This ability is ensured by connectors, responsible for data flow and event transfer. The corresponding BIP code for obstacle location in Fig.5 and the main driving component in Fig.6 are portrayed in appendix B.1 and appendix B.2, respectively.

Listing 1.1: A partial self-driving vehicle in BIP: Servers location

```
// Package declaration
package selfDriving
 // External function declaration
 extern double function findClosestServers( )
 // Atom declaration
 atom type Servers_location()
  // Data declaration
  data double delay
  // Exporting ports
  export port Port_t trigger_server_location()
  export port Port_t servers_located()
  // Atom states declaration
  place START, SERVERS_LOCATED
  initial to START  // trigger the initial state
  // Synchronized transitions using ports
  on trigger_server_location
    from START to SERVER_LOCATED do
    {delay:= findClosestServers();}
  on servers_located
    from SERVER_LOCATED to START
 end  // end atom declaration
```

States denote control locations where components wait for interactions. A transition is an execution step, labeled by a port, from one control location to another. Each transition has an associated guard and action (s). For instance, the transition leaving the state *Obstacle_Located* to *Servers_Location_Triggered* in the main vehicle behaviour in Fig.6 is fired only if the obstacle is detected (i.e. the function isObstacleInFront() returns true/false), else, a transition is fired leading to the state *Acceleration*. After servers have been located, the vehicle checks if the overtaking is enabled (i.e. the function isOvertakingEnabled() returns true/false), in this case, the acceleration state is enabled, else deceleration is enabled. From Acceleration/Deceleration states, the acceleration is updated, and *Position_Changed* state is enabled. The transition leading to $Velocity\_Regulation$ updates the vehicle position and its velocity according to the equations Eq.2 and Eq.3.

5 Analysis using BIP toolchain

Fig.7 portrays the roadmap from specification to analysis and code generation. Building and verifying systems in BIP requires three inputs: (i) the architecture in BIP language, (ii) a low-level code in C++ to communicate with devices those related to perception and decision making, (iii) properties expressed in PBLTL.

The BIP compiler is responsible for reading user input (i.e. BIP source code) through a parser and produces the final result in C++. Finally, composed source code is produced by gluing the external C++ files with the transformed ones. An executable artifact is generated for simulation or to be embedded on a specific platform.

The generated code could be feed to the stochastic engine where almost probabilistic models are covered such as Markov Decision Process (MDP), Discrete-time Markov Chain (DTMC), and CTMC. The stochastic engine encapsulates an executable model simulator and it is used in order to produce (random) execution traces on demand. The monitor is used to evaluate properties on traces, and, then to produce local verdicts $\{true, false\}$. Moreover, the SMC engine implements the main statistical model checking loop depending on the statistical method used, namely, hypothesis testing [30] or probability estimation [11]. Finally, the parametric exploration module coordinates the evaluation of a parametric property. Also, the tool[1] offers an integrated development environment including a graphical user-interface permitting to edit, compile, simulate models, and plotting graph for parametric properties. To perform verification and simulation, experimentations are run on Ubuntu-18.04 machine with Intel Core i5-4310U@2.00GHz and 16 GB RAM.

5.1 Verification using SMC-BIP

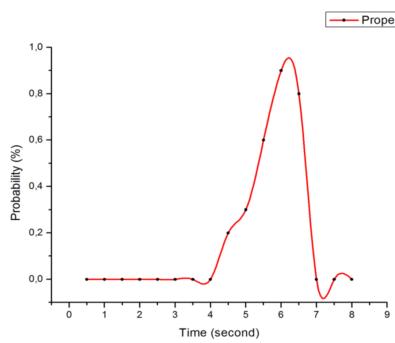

Fig. 8: SMC results for the property 2.

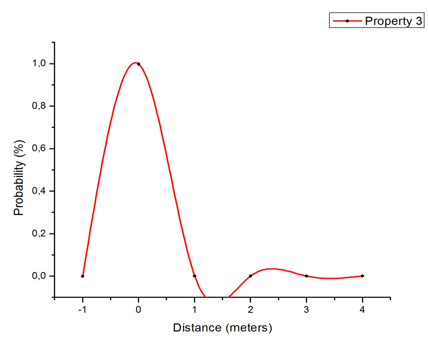

Fig. 9: SMC results for the property 3.

The properties specification language over stochastic systems is a probabilistic variant bounded Linear-time Temporal Logic (LTL). Using this language, it is possible to formulate two kinds of queries on the given system:

- Qualitative queries : $P_{\geq\theta}[\varphi]$, where $\theta \in [0, 1]$ is a probability threshold and φ is a bounded LTL formula.
- Quantitative queries : $P_{=?}[\varphi]$ where φ is a bounded LTL formula.

Path formulas are defined using four bounded temporal operators namely, `Next` $(N\psi_1)$, `Until` $(\psi_1 \cup^k \psi_2)$, `Eventually` $(F^k\psi_1)$, and `Always` $(G^k\psi_1)$, where k is an integer value that specifies the length of the considered system execution trace and ψ_1,ψ_2 are called state formulas, which is a Boolean predicate evaluated on the system states.

[1] http://www-verimag.imag.fr/BIP-SMC-A-Statistical-Model-Checking.html?lang=en

We use the SMC-BIP tool with the confidence parameters $\alpha = 0.005$ and $\delta = 0.05$ for all our experiments. For example, the LTL formula:

$$P_{=?}[F^{1000}(X_{next} = 300 \ \&\& \ elapsed\_time = 10 \ \&\& \ V_{next} = 10)] \tag{1}$$

answers the question " *What is the probability of having the traveled distance is equal to 300 meters in the next* `10` *seconds with velocity equal to* `10` *m/s?* " The query is satisfied, and the returned probability is `0.80`. Considering this probability, designers judge the quality of the results and may regulate certain parameters if it does not respond to the requirements.

The second query is related to the verification that the self-driving car will change the lane if and only if the overtaking is enabled. The elapsed time shall be at least `6.17` *seconds* if the acceleration is equal to `1.25` m/sec^2. The corresponding PBLTL property is expressed as follow:

$$P_{=?}[c\_lane = false \ \cup^{1000} \ (c\_lane = true \ \&\& \ elapsed\_time \leq T \ \&\& \ a = ACC \ )] \tag{2}$$

where T =6.17 refers to the maximum time for overtaking according to the motion equations, and ACC=`1.25`. The confidence parameters cited above require the evaluation of 1199 system executions to come up with a global verdict, using the probability estimation technique. The graph in Fig.8 resulting from the statistical model checking of property 2 portrays the probability evolution for the self-driving car reaching `98%` to achieve the overtaking after 6 seconds have been elapsed and less before 6 seconds. It is clear that the meaning of the maximum probability is to fit the basic acquaintance on the overtaking.

Supposing that the obstacle is detected and we have to ensure that the self-driving vehicle keeps a stable distance of `4.0` *meters* from the truck in front when no overtaking is possible. The according PBLTL property is expressed in the form of:

$$P_{=?}[(obstacle = true \ \&\& \ distance = D) \ \cup^{1000} \ (obstacle = true \ \&\& \ distance = D + d)] \tag{3}$$

where d is an integer value ranging from `-1` to `4` with increment equal to 1. The graph in Fig. 9 portrays the evolution of the probability regarding the distance incrementation. It is visible that the safety distance is respected when the overtaking is not possible while reaching the maximum probability to `99.8%`.

5.2 Simulation

During the simulation, we record the self-driving vehicle acceleration and velocity. Also, we record the number of times that the vehicle performs the overtaking when an obstacle is detected. The testbed to perform such simulation is portrayed in Fig.10. The architecture is organized in two layers. The first layer constitutes the C++ generated code from BIP as portrayed in Fig.7. The second layer represents the main function to perform overtaking such as servers location and obstacle detection. The communication requests/responses between the two layers are achieved by JSON files. The second layer is build using python language where functions of that layer are externally exposed through Flask server. For experimentation, the trained labeled images dataset is collected from Udacity self-driving project [24]. Both layers are deployed over a Raspberry PI 4 with 1GB RAM.

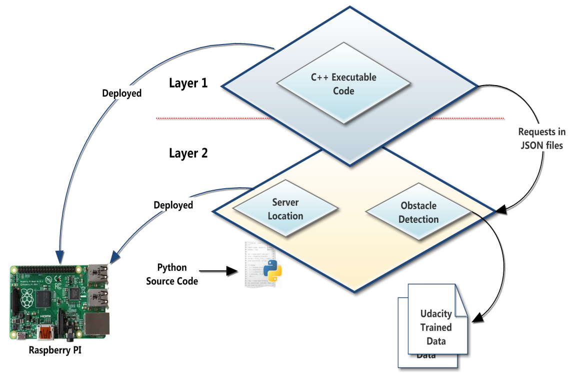

Fig. 10: Vertical architecture of the simulation platform.

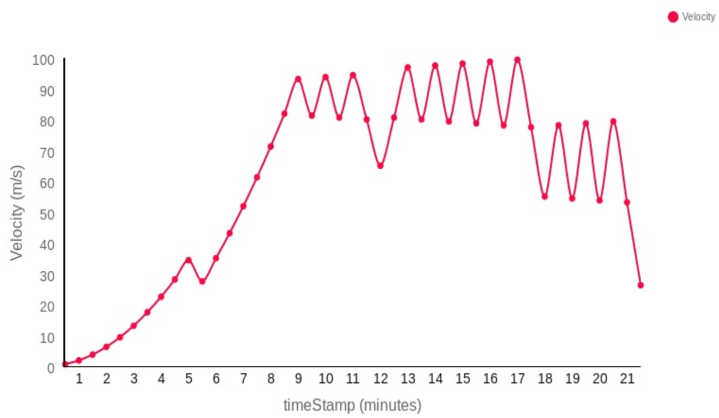

Fig. 11: Self-driving vehicle recorded velocity.

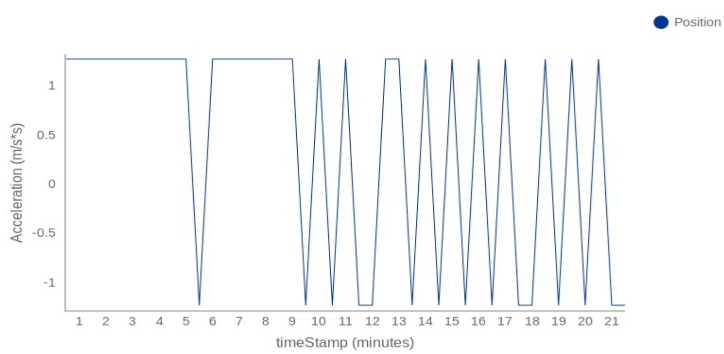

Fig. 12: Self-driving vehicle recorded acceleration.

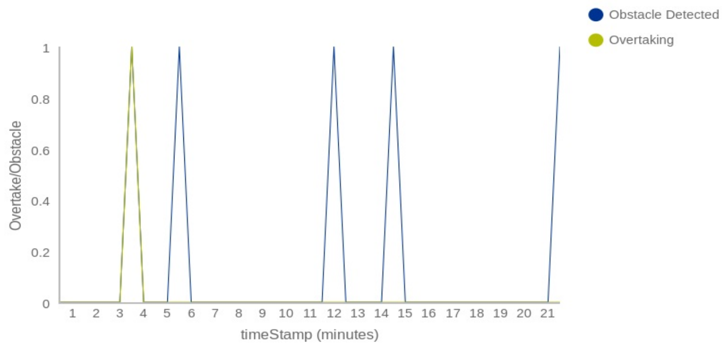

Fig. 13: Self-driving vehicle overtaking and obstacle detection.

In Fig.11, we observe the evolution of the velocity regarding the time progress (i.e. minutes). The self-vehicle attains `90 m/s` after 9 minutes have been passed and then the velocity ebb and flow stays between `80 m/s` and `90 m/s` until the 12^{th} minutes, where the vehicle decreases its velocity to `65 m/s` when obstacle is detected and due the closest distance to it. Meanwhile, before attaining the arriving position, the vehicle decreases its velocity for smooth braking.

In Fig.12, we observe the evolution of the acceleration regarding the time progress. It is important to note that during the simulation, the acceleration is within the interval $[-1.25, 1.25]$. During the overtaking, the acceleration should be at its maximum; however, when acceleration is impossible, its value stays at its minimum. For instance, at the 12^{th} minutes, the self-driving car starts accelerating despite a detected obstacle in front (Fig.13). The overtaking was possible because the assisted servers returned undetected cars around the self-driving vehicle. Meanwhile, from the 14.5 minutes the self-driving vehicle performs deceleration because in the highway a maximum allowed velocity is `100 m/s` and it totally in accordance with the portrayed automata that the self-driving car shall fit the constraint `Vnext` $\geq$ `SafeVelocity()`.

6 Conclusion

In this paper, we have presented an integrated approach for building and automatically generating software code for self-driving vehicles satisfying various requirements. The overtaking assistance is enabled by an auxiliary sensing system that will endow the self-driving car with capabilities that are not available onboard electronic systems. The majority of the computation is performed over a virtual simulator while collecting the information from its surrounding environment. We verify the models in BIP describing the overtaking against PBLTL queries expressing self-driving vehicle requirements essential to their respective missions. Also, the statistical model checking proves that the system is faithful to those values obtained during the simulation.

Future work has at least two potential directions. One is to combine statistical model checking with machine learning to improve the efficiency of searching through the generated trace log. Another direction is related to the integration of more decision-making components within the architecture to observe and measure the dynamic reactivity to the obstacle detection.

References

1. Baidu, apollo,, http://apollo.auto/
2. Abomhara, M., Køien, G.M.: Security and privacy in the internet of things: Current status and open issues. In: 2014 International Conference on Privacy and Security in Mobile Systems (PRISMS). pp. 1–8 (May 2014)
3. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. **28**(1), 6:1–6:39 (jan 2018). https://doi.org/10.1145/3158668
4. Ballarini, P., Barbot, B., Duflot, M., Haddad, S., Pekergin, N.: Hasl: A new approach for performance evaluation and model checking from concepts to experimentation. Performance Evaluation **90**, 53 – 77 (2015)
5. 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 model checking. In: 2019 IEEE Intelligent Vehicles Symposium (IV). pp. 252–259 (2019)
6. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the bip framework. IEEE Software **28**(3), 41–48 (May 2011)
7. Charlton, J., Gonzalez, L.R.M., Maddock, S., Richmond, P.: Simulating Crowds and Autonomous Vehicles, pp. 129–143. Springer Berlin Heidelberg, Berlin, Heidelberg (2020)
8. COSMOS: Cosmos tool. http://www.lsv.ens-cachan.fr/software/cosmos/ (2015), http://www.lsv.ens-cachan.fr/Software/cosmos/
9. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for biological systems. Int. J. Softw. Tools Technol. Transf. **17**(3), 351–367 (Jun 2015). https://doi.org/10.1007/s10009-014-0323-4
10. David, A., Larsen, K.G., Legay, A., Mikuăionis, M., Poulsen, D.B.: Uppaal smc tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (Aug 2015). https://doi.org/10.1007/s10009-014-0361-y, https://doi.org/10.1007/s10009-014-0361-y
11. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Verification, Model Checking, and Abstract Interpretation. pp. 73–84. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
12. Houenou, A., Bonnifait, P., Cherfaoui, V., Wen, Y.: Vehicle trajectory prediction based on motion model and maneuver recognition. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4363–4369 (2013)
13. Jeong, N., Hwang, H., Matson, E.T.: Evaluation of low-cost lidar sensor for application in indoor uav navigation. In: 2018 IEEE Sensors Applications Symposium (SAS). pp. 1–5 (2018)
14. MRMC: Mrmc tool. http://www.mrmc-tool.org (2011), http://www.mrmc-tool.org
15. Ngai, D.C.K., Yung, N.H.C.: A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers. IEEE Transactions on Intelligent Transportation Systems **12**(2), 509–522 (2011)
16. Nvidea: Nvidia drive agx, https://www.nvidia.com/fr-fr/self-driving-cars/drive-platform/hardware/
17. Organization, W.H.: Road traffic injuries (7 February 2020), https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries
18. Paden, B., Cap, M., Yong, S.Z., Yershov, D., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on Intelligent Vehicles **1**(1), 33–55 (2016)

19. Petrov, P., Nashashibi, F.: Modeling and nonlinear adaptive control for autonomous vehicle overtaking. IEEE Transactions on Intelligent Transportation Systems **15**(4), 1643–1656 (2014)
20. Prabhakar, G., Kailath, B., Natarajan, S., Kumar, R.: Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving. In: 2017 IEEE Region 10 Symposium (TENSYMP). pp. 1–6 (2017)
21. Prati, M.V., Costagliola, M.A., Pagliara, F., Mastantuono, E.: Idling vehicle emissions and fuel consumption in urban use: Influence of the stop start technology. In: 2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe). pp. 1–4 (2018)
22. Ramanagopal, M.S., Anderson, C., Vasudevan, R., Vasudevan, R., Johnson-Roberson, M.: Failing to learn: Autonomously identifying perception failures for self-driving cars. IEEE Robotics and Automation Letters **3**(4), 3860–3867 (2018)
23. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 779–788 (2016)
24. Udacity: Annotated driving dataset, https://github.com/udacity/self-driving-car/tree/master/annotations
25. Wang, X., Wei, T., Kong, L., He, L., Wu, F., Chen, G.: Ecass: Edge computing based auxiliary sensing system for self-driving vehicles. Journal of Systems Architecture **97**, 258 – 268 (2019)
26. Xiao, W., Zhang, L., Meng, D.: Vehicle trajectory prediction based on motion model and maneuver model fusion with interactive multiple models. In: WCX SAE World Congress Experience. SAE International (apr 2020)
27. Yang, C., rong Shao, H.: Wifi-based indoor positioning. IEEE Communications Magazine **53**(3), 150–157 (2015)
28. Yoganandhan, A., Subhash, S., Hebinson Jothi, J., Mohanavel, V.: Fundamentals and development of self-driving cars. Materials Today: Proceedings (2020)
29. Younes, H.L.S.: Ymer: A statistical model checker. In: Computer Aided Verification. pp. 429–433. Springer Berlin Heidelberg (2005)
30. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification. pp. 223–235. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
31. Yurtsever, E., Lambert, J., Carballo, A., Takeda, K.: A survey of autonomous driving: Common practices and emerging technologies. IEEE Access **8**, 58443–58469 (2020)

A Server detection algorithm

As demonstrated in section.3, servers within the communication coverage of self-driving vehicles will send information including their ID and localization to the self-driving vehicle. The vehicle will select two nearest servers on the same side to assist accurate vehicle localization as detailed in Algorithm.1.

Algorithm 1: The server selection algorithm

Data: $(x_{v,a}, y_{v,a})$: the position of the vehicle a; S: the set of servers that are in the communication range of the vehicle a; N: the number of servers in set S; S_i: server i; $(x_{s,i}, y_{s,i})$: the position of server i

Result: S_{pre}: the set of communcation base stations

$d_{min} \leftarrow \sqrt{(x_{v,a} - x_{s,1})^2 + (y_{v,a} - y_{s,1})^2}$;
$S_{pre} \leftarrow \emptyset$;
$m \leftarrow 0$;
$k \leftarrow 0$;
while *(j ¡ N)* **do**
  $d \leftarrow \sqrt{(x_{v,a} - x_{s,j})^2 + (y_{v,a} - y_{s,j})^2}$;
  if $(d < d_{min})$ **then**
    $d_{min} \leftarrow d$;
    $k \leftarrow m$;
    $m \leftarrow j$;
  end
  $j \leftarrow j + 1$;
end
$S_{pre} \leftarrow S_{pre} \cup S_m$;
$S_{pre} \leftarrow S_{pre} \cup S_k$;
return S_{pre};

B BIP code of the case study

B.1 BIP code of obstacle detection

Listing 1.2: A partial self-driving vehicle in BIP: Obstacle location

```
extern boolean function isObstacleInFront( )
extern boolean function getObstacleLocation( )
atom type Obstacles_location()
 data boolean obstacle
 data double distance
 export port Port_t trigger_obstacle_location(
    obstacle)
 export port Port_t obstacle_located()
 place START, OBSTACLE_LOCATED
 initial to START  // trigger the initial state
  on trigger_obstacle_location
   from START to OBSTACLE_LOCATED do
   {obstacle:= isObstacleInFront( );
    distance:=getObstacleLocation( );}
 on obstacle_located
   from OBSTACLE_LOCATED to START
end  // end atom declaration
```

B.2 BIP code of the self-driving

Listing 1.3: A partial self-driving vehicle in BIP: Main component

```
extern double function overtakingAcceleration( )
extern boolean function isOvertakingEnabled ( )
extern int function changeLane  ( )
extern double function overtakingDeceleration()
extern double function SafeVelocity(()
atom type Main()
 data boolean obstacle, c_lane
 data double a, Xpast, Vpast, Xnext, Vnext, delta,
     elapsed_time
 export port Port_Value trigger_obstacle_location(
     obstacle)
 export port Port_t obstacle_located()
 port Port_t silent_p3()  // port silent
 export port Port_t trigger_server_location()
 export port Port_t servers_located()
 place START, OBSTACLE_LOCATED,
     OBSTACLE_DETECTION_TRIGGERED
 initial to START  // trigger the initial state
 // Synchronized transitions using ports
 on trigger_obstacle_location
   from START to OBSTACLE_DETECTION_TRIGGERED
 on obstacles_located  from
     OBSTACLE_DETECTION_TRIGGERED to
     OBSTACLE_LOCATED
 on trigger_server_location  from OBSTACLE_LOCATED
     to SERVERS_LOCATION_TRIGGERED provided (
     obstacle==true)
 on silent_p from SERVERS_LOCATED to ACCELERATION
     provided (isOvertakingEnabled()==true)
 .....
 on silent_p3 from POSITION_CHANGED to
     VELOCITY_REGULATION do{
 Xpast=Xnext;    Vpast=Vnext;
 Xnext=Xpast+Vpast*delta+0.5*a*delta*delta;
 Vnext=Vpast+a*delta;
 elapsed_time= elapsed_time+ delta;
 c_lane=false;   }
 on silent_p4 VELOCITY_REGULATION  to start
     provided (Vnext<SafeVelocity())
end  // end atom declaration
```