A Case Study on Formally Validating Motion Rules for Autonomous Cars

Mario Henrique Cruz Torres, Jean-Pierre Giacalone, Joelle Abou Faysal

A Case Study on Formally Validating Motion Rules for Autonomous Cars

Mario Henrique Cruz Torres[1], Jean-Pierre Giacalone[2], and Joelle Abou Faysal[3]

[1] IVEX.ai Intelligent Vehicle Technology, Leuven, Belgium
[2] Renault SW Labs (RSL), Expert ADAS/AD software architecture, Autonomous Vehicle Algorithms, Sophia Antipolis, France
[3] Renault Software Labs (RSL), Université Cote d'Azur, Cnrs, Inria, I3S, Researcher, France

Abstract. Car motion control is a key functional stage for providing advanced assisted or autonomous driving capabilities to vehicles. Car motion is subject to strict safety rules which are normally expressed in natural language. As such, these natural language rules are subject to potential misinterpretation during the implementation phase of the motion control stage. In this paper, we show a novel approach by which safety rules are expressed in natural language, then in a formal language specification which is then validated and used to generate a car motion checker. We present a case study of using the approach with true road capture data and its associated imperfections. We also show how the approach lowers the validation efforts needed to guarantee that the car motion always respects a desired set of safety rules while other traditional validation methods would be much heavier to deploy and error prone.

Keywords: Formal Language · Autonomous Drive · Motion Safety

1 Introduction

In the past few years, across most regions in the world, there has been a push to improve vehicles driving safety through specific regulations. In Europe, for instance, this has been the case with the issuance of General Safety Regulation (GSR) phase 2 [6]. These regulations tend to mandate the deployment of Advanced Driving Assistance systems in cars, like Automatic Emergency Braking and other car motion control systems such as Autonomous Emergency Steering or Adaptive Cruise Control [5]. These driving features take control of car motion on behalf of the driver, even if the driver still has the ability to take back control, in order to operate within time ranges and with perception reaction time that allows fast action to protect the car occupants. As such, these systems must make sure that passengers safety is guaranteed during their operation.

For such systems, a key element to guarantee safety is the car trajectory control. The trajectory control component is the part that computes the future trajectory of the vehicle as a function of what is perceived by the car sensors. The trajectory is transformed into 2D positional objectives to follow on the

road, from the current position. The problem behind ensuring safety at the current position and along the trajectory is bounded in terms of the expression of the safety rules. Important properties behind such bounding are related to creating priorities (e.g., what is happening at the front of the vehicle versus at the rear) or having parallel conditions expressed (e.g., checks being performed simultaneously on different directions, longitudinally and laterally) [15]. Hence, car motion safety construction is a perfect domain for exploring formal methods to express safety rules.

In the context of proof that is needed to express safety rules described above, one can refer to several specification languages providing design by contract approaches ([1], [12] for instance). Usually, though, these languages are not at the right level of abstraction to express key aspects regarding car motion. In this paper, we will explain what that level should be and why. As it will be shown, there are several challenges expressing safety rules about motion control. As a result, we will describe the expected flow to be used between the expression of rules in human language by a car safety engineer and their formalization using the proposed approach. We will also illustrate the application of the formal method depicted in this paper through true road captures of difficult scenarios in traffic jams, with complicated perception situations (at night, for instance). We will devise the impact of those conditions as constraints in the expression of safety rules. Finally, we will indicate some trends in the application of this technique for safe car motion control implementation.

This paper is organized as follows. Section 1.1 briefly discusses related work. Section 2 presents the challenges in implementing safety rules concerning car motion control. Section 3 introduces IVEX tools and details IVEX approach to model safety rules for car motion. Section 4 describes a case study of a Society of Automotive Engineering (SAE) Level 3 car experiments. It also discusses the low speed motion control safety rules set. Finally, Section 5 draws conclusions and details some possible future work.

1.1 Related Work

Reachability analysis is used to propose a safety framework to analyse the motion of autonomous vehicles by used at [13]. It is used to guarantee that any feasible future motion of dynamic obstacles around the automated car, also known as Ego car, is taken into account when assessing a planned trajectory. Similar to our technique, [13] define a set of assumptions for the future movement of dynamic obstacles around the Ego car and verifies if the Ego car can reach a future state where it would have a collision. An interesting aspect of the the work [13] is to also provide an alternative fail-safe trajectory planning that the autonomous vehicle (AV) could use to avoid a collision.

The work by [16] proposes a mathematical model, called Responsibility-Sensitive Safety (RSS), for safety assurance of autonomous vehicles. The RSS model is explainable and has well defined assumptions. The model tries to formalize common sense human behavior concerning the judgment of "who is re-

sponsible for causing an accident". The final goal of the model is to guarantee that an autonomous car never causes an accident, being than considered safe. The main limitation of the model is not taking variability and uncertainty into account. This limitation greatly impacts the ability of RSS to guarantee safety in reality.

2 Challenges in Implementing Safety Rules around Car Motion Control

When creating safety rules for a car motion control we assume that it is impossible to guarantee there is absolutely no collisions involving the controlled car. The impossibility of zero collisions is due to the fact that the Autonomous Vehicle/Advanced Driver Assistant Systems (AV/ADAS) car cannot control the behaviour of other road users. If another road user is actively trying to cause a collision and has a vehicle capable of very large accelerations/decelerations, it is easy to understand that even if the AV/ADAS executes evasive maneuvers or stops on the side of the road, the other road user can still cause a collision. There is a trade-off between the drive-ability and safety of the controlled car, since there will always be a risk of a collision. We believe that creating the safety rules for the car motion control will always incorporate such trade-off.

The first challenge for creating safety rules for car motion control is identifying the minimum set of assumptions (including hidden assumptions) about the environment, particularly assumptions about other road users, which are measurable and represent reality closely enough. The RSS model [16], for instance, defines a small number of assumptions about the environment, such as the maximum, minimum acceleration/deceleration of other vehicles which leads to a model that can be easily understood by human beings but which also leads to lower drive-ability of a controlled car. For instance, one's assumptions for maximum deceleration may lead to an extremely conservative driving behaviour, thus lowering the drive-ability of the controlled car.

We believe the car motion safety rules should be formally verified for soundness and completeness for a certain environment. This brings the second challenge which is defining the proper abstraction level to formally model the safety rules. Each formalism requires the system (software + hardware) to be specified in a certain way, [9] [17]. The modeler has to reduce, simplify, or abstract the system being modeled to be able to use different model checking tools.

Formally modelling the system and its environment has to be done taking into account possible issues, such as:

- the model does not represent the system,
- the model does not properly represent the environment (e.g. wrong assumptions about the environment),
- the model truly represents the system and its environment, but is intractable.

Finding the correct level of abstraction to represent the AV/ADAS system and its environment is challenging because it has to be done in a way that the

model is close enough to the reality, but abstract enough to be solvable [7]. Toolboxes like Tulip [18] help to mitigate the problem of having a model which does not represent the system, since it synthesizes controllers, but does not help in defining the proper abstraction level to represent the problem, or mitigating having wrong assumptions about the environment.

Particular attention has to be given to modeling the input information that will be used by the motion control rules. The perception systems which provide information used by the motion controllers may have a great impact into the safety rules. When modeling the car motion rules, it is extremely important to clearly model the assumptions concerning the perception systems used in the car so that limitations of this system can be properly dealt with in the motion of the car.

The focus of this paper is on car motion control under conditions of low speed (lower than 60 km/h) and traffic jams. The control context is either Advanced Driving Assistance where the driver is still under control but is assisted by the electronic system or Autonomous Driving of Level 3, as defined by the SAE [14]. Motion control consists in constructing a trajectory to be followed by the car under control. This trajectory is expressed as 2D positions on the road, provided with a recurrence of 20 to 100 ms into the future from current time. The positions are expressed in the car coordinates (see Fig. 1). Car motion control takes these target positions and transforms them into actions along the longitudinal direction (along the X axis) and the lateral (along the Y axis through the yaw angle) one, for the vehicle. Simply put, these actions relate to defining acceleration or deceleration of the car and steering wheel movement. As a result, there will be rules for checking longitudinal and lateral safety, and these rules are going to be valid simultaneously which is another challenge we have to address with the rules description language.

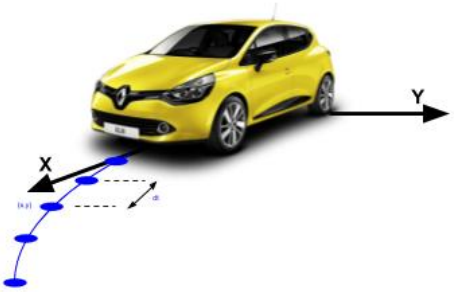

Fig. 1. Car motion trajectory definition in the car coordinates system.

Car motion control constructs the trajectory based on information reported by sensors mounted in the vehicle. This information is usually named Perception and consists in aggregating different details about moving or static objects around the car, like type, dimensions, position, speed and acceleration. This ag-

gregation is performed by an electronic system called Fusion that materializes and confirms the various detections provided by individual sensors. Depending on the number and type of sensors available, the accuracy and reliability of the information may vary. Challenges regarding obtaining a quality Perception have been highlighted in publications like [8]. Among other issues related to perception conditions (night, rain, fog, as examples), problems of persistence are quite impacting to the definition of safety rules like the ones introduced above. In essence, potential appearance and disappearance of detected obstacles means that safety rules must express a dependency in space (longitudinal, lateral) and time (provided an obstacle is confirmed over a certain time, for instance). And this becomes a constraint to the description language.

As we will see in section 4, the approach for implementing car motion safety rules presented in this paper has been exposed to real road data captured with a car prototype embedding several classes of sensors (cameras, radars, lidars, ultra sonic) and called TRAJAM. We will see clear examples of perception challenges that were faced in order to properly express rules given by a human safety engineer.

3 IVEX Tools Suite and Approach to Model Safety Rules for Car Motion

The IVEX toolchain can be used to model different systems that need safety guarantees and which operate in complex environments. IVEX engineers spent years performing research into the development of safety critical systems for other domains, such as aerial vehicles and Automated Guided Vehicles (AGVs) [3] [2] [4]. When doing research, they modelled different systems using varied approaches, exploring diverse ways to specify safety critical autonomous systems. IVEX engineers understood that traditional approaches like creating Finite State Machines (FSM), Behavioral Trees, and traditional formal modeling techniques, like solvers for LTL, had their own limitations to define safe autonomous systems [11]. The systems build by IVEX engineers were used and demonstrated, besides others, in aerial platforms having embedded mission control and autonomous safe behavior, used to fly around electricity towers in Belgium.

At the core of the IVEX toolchain is an engineering process (Depicted in Figure: 2) which supports the creation of the safety rules. The process allows one to automatically transform safety requirements into formally verified software. The toolchain then generates correct-by-construction software, by performing a translation between a solved model specification and a C++ execution policy. The process highlights the limitations of safety requirements (by performing consistency and completeness checks). The process shortens iteration cycles, reuses existing knowledge and is supported by mature toolchain.

The first step into the process is to identify the safety requirements that should be always satisfied by the car motion. Normally, such requirements reflect a number of safety requirements imposed on the car motion. The safety

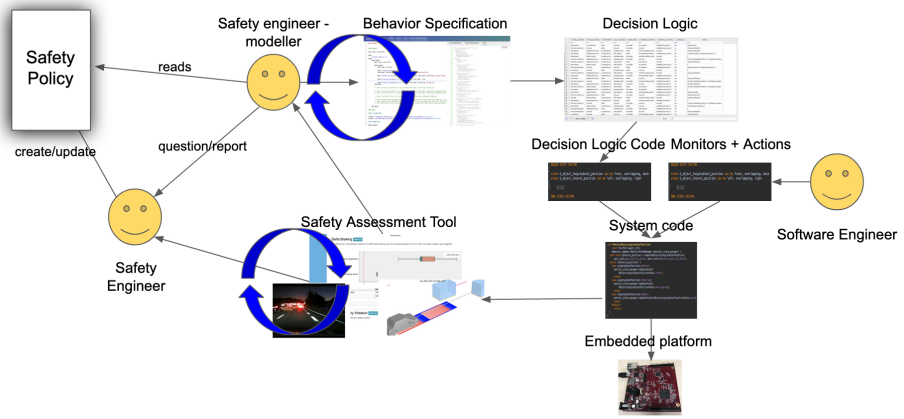

Fig. 2. The IVEX process consists of 3 main steps: 1 - define a behavior specification of the system and its environment, 2 - generation of a decision making logic, 3 - validation of the created system using the Safety Assessment Tool.

requirements can even consider what is the expected car behavior in different operational design domains (ODD).

Based on the requirements gathering on system goals, safety and other (as gathered together with the various system stakeholders), IVEX engineers specify the requirements in a behaviour specification. During specification, a concise, formal model of the system is built, including of the perceivable system states, its actuation, rules and constraints that must be fulfilled. The behavior specification is created using a domain specific language (DSL). The DSL has constructs to represent vehicle system properties, such as pre-conditions for action executions, and expected outcomes. The behavior specification is written in the DSL using first-order logic constructs. The specification is declarative (it describes goals, state, actions - as in a traditional planning model - and constraints), it is not imperative (describing for every situation exactly which action to take). This is a fundamental aspect of the approach, which makes it more adaptable and manageable from the ground up. The exact decision making logic is generated later on in the process.

The next step is performing an automated analysis on the specification, by using IVEX verification tool, to check whether the specification is:

- complete (the specification will be able to decide in each possible state)
- consistent (the specification does not contain contradicting requirements)

It could be that (a) the specifications is not covering a combination of states; or (b) that the model is inconsistent. The verification tool will automatically detect and report inconsistencies. If a specification is not verifiable, developers receive clear and punctual feedback on the status of the specification. Through iterative specification and verification steps, developers are guided to unambigu-

ously and completely model the expected system behavior under all circumstances.

Based on the final, verified specification, the toolchain generates the decision making logic as code. At this step the decision logic is translated into a tree-like data structure with a known maximum depth, which is critical for guaranteeing real-time execution deadlines. Besides that, the decision logic is a direct mapping of the behavior specification into C++ code, which lowers the chances of implementation bugs.

The logic implements a mapping for every possible discrete situation - based on the state representation from the requirements & specification - to one or a set of actions to be executed by the system. This logic is guaranteed to cover every possible discrete scenario, and respect all safety requirements and other constraints. A typical specification for a SAE Level 3 car generates around 120.000 safety rules in the decision logic, when considering many aspects of the Operating Design Domain.

The toolchain has a runtime execution environment which is used to perform the integration of the execution policy with the rest of the system, called the IVEX safety co-pilot. The runtime includes specific components for integration in the overall autonomous system. The runtime has well defined inputs and outputs interfaces, to facilitate its integration into the system. For instance:

- The runtime communicates with the rest of the system via a middleware or via direct function invocation (loading new threads to execute continuous controllers).
- Inputs are read by **monitors**. Monitors are runtime components that actively read the perception data from the system and convert this perception data into discrete values. Example of monitor inputs are: static and dynamic obstacle location (with corresponding confidence levels), car velocity, etc. One of the inputs for car motion validation is the planned trajectory created by a path planner.
- Outputs are given by **actions**. Actions represent the actuator components from the autonomous system that should be activated and their parameters. For instance, an action can represent an emergency operation such as a strong longitudinal braking controller.

The process has two distinct parts, being an off-line one, used to model the system, and an on-line one used to verify the behaviour of the system during execution. In order to validate the full system that uses the decision making logic the process has a validation step. The first validation step is done by testing the created system with recorded driving data in the IVEX Safety Assessment Tool (SAT). The SAT tool performs Software-in-the-loop (SIL) tests, using the system created.

The SAT tool allows one to replay recorded driving sensor data into the SIL which will then check its thousands of decision rules to define if a certain car movement is triggering a safety violation. The SAT tool then collects all safety violations occurrences and generates statistics highlighting all critical situations in the driving data. A safety engineer can then proceed to analyse the highlighted

safety violations and the safety metrics created by the SAT tool. After analysing thousands of scenarios, the SAT tool indicates how conservative the created system is, allowing safety engineers to proceed to refine the behavior specification or its parameters.

4 Case Study of a SAE Level 3, Low Speed Motion Control Safety Rules Set

In this section, we are entering into real experiments conducted using the tools and method described in the previous section through information available in a re-simulation environment. A re-simulation environment is providing data captured during road trips by a car embedding a set of sensors close to the one used in production and located where they would be installed. Hence, study presented here is based on real data. The environment provides data at various locations in the processing stages through pipes that can be connected to the system to be tested. In our case, these pipes carried kinematic information from sensor fusion outputs for objects, infrastructure and Ego (a.k.a. the automated) car. They also carried the future Ego car trajectory positions as delivered by the planning stage. The infrastructure data consisted of lanes structure information as captured by the perception stage. Kinematic information included positions in 2D as measured in Ego car referential (see Fig. 1) as well as speed and acceleration. The re-simulation environment also provides a situational camera view, towards the front of Ego car, synchronous to the provided data in order to better understand visually a given scenario configuration. Table 1 summarizes the re-simulation data available.

Table 1. Data available through re-simulation.

Type	Content	Sampling
Objects	Position, Speed, Accel., Size	40 ms
Ego car	Position, Speed, Accel.	40 ms
Infrastructure	Lines types, Shape	40 ms
Trajectory	Positions	100 ms

With this re-simulation environment, we constructed and verified car motion safety rules corresponding to the SAE Level 3 motion control mode. These were written as a real safety policy, describing longitudinal and lateral situations to be avoided and corresponding expected behavior. Fig. 3 shows how these rules are getting exercised with the re-simulation data. The Trajectory Validation function receives the results of the analysis according to safety rules, and apply them on the trajectory data proposed by the planner. This valid trajectory will be passed to the motion control that follows it (as explained earlier). In case checks report a failure in fulfilling the rules then an emergency maneuver could be signaled by this function, as an example.

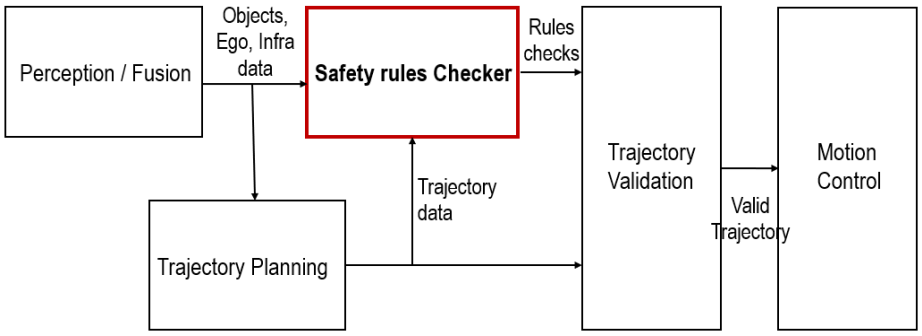

Fig. 3. Safety rules verification high level architecture. Each rectangle represents a functional component in the system. The safety rules are checked at the **Safety rules Checker** component which receives information from the **Perception/Fusion** and **Trajectory Planner** components. The results of the checks performed by the **Safety rules Checker** components are sent to the **Trajectory Validation** component which is then responsible for deciding on following the planned trajectory or not.

Safety rules considered here were initially structured and expressed in human text language. Situations covered for L3 control mode were essentially in traffic jam, at low speed and various weather and light conditions as well as infrastructure and slopes. Organized in tables, the rules provide information about the situation being verified, the preconditions, the result being avoided (usually, a collision) and specific aspects to consider. Their definition is owned by a safety engineer from Renault and he was supporting requests for understanding situations to be checked in case there was any ambiguity. An example of longitudinal safety description is depicted in Fig. 4. There were around 20 rules like this that composed the L3 set.

By going over this rule, we can first observe that a traffic jam situation must be the operating case (also known as the Operating Design Domain or ODD). This means Ego car is surrounded by several moving objects. We can also observe that it is following a preceding vehicle with which a safety distance of 2 seconds is defined. The rule sets a situation by which the preceding vehicle is potentially decelerating with a certain strength (minimal for $1m/s^2$, nominal for $5m/s^2$ or strong for $10m/s^2$). This covers limits of the environment model as discussed in section 2. On his side, the Ego car has the capability to regulate its speed with an Automatic Cruise Control (ACC) deceleration capability up to a strong braking capability of 10 m/s2. In order to model this rule, we need to start discretizing space to separate system states with the preceding vehicle. Fig. 5 describes this, based on the rule content. The blue vehicle is Ego car.

Longitudinal states are so that either Ego car is alone or there is a preceding vehicle. And, if there is one, then the situation can be that Ego car is at a safe distance, i.e. 2 seconds from it, or it is within a range of ACC distance where it should regulate with corresponding deceleration levels or it is within an emer-

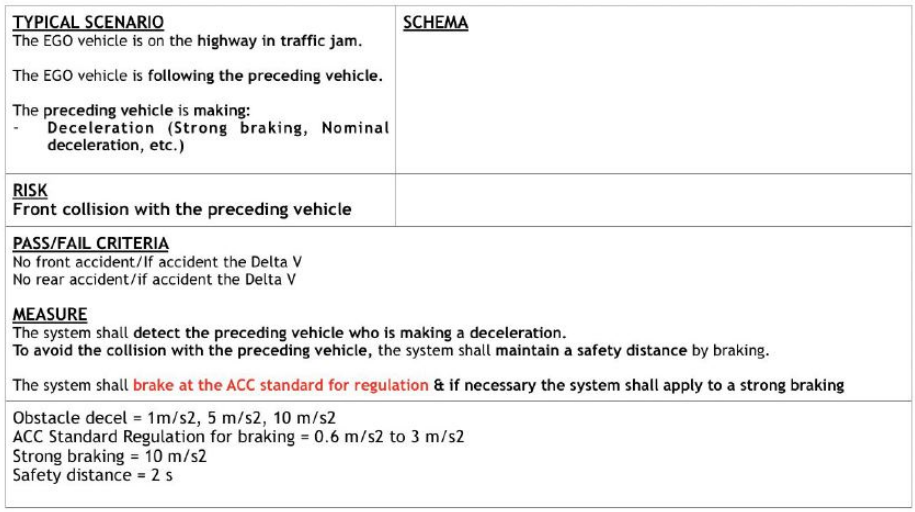

Fig. 4. Longitudinal rule expressed by safety engineering.

gency distance that requires to regulate speed with strong braking capabilities. The resource that is actionable is longitudinal acceleration. The rules brings up 2 system states that matter for verifying it: The traffic jam state (S_traffic_jam, a Boolean, yes/no) and the front car distance state (S_front_car, an enumerated, not_exist, safe_distance, acc_distance, emergency_distance). Each state values are provided by monitors that run at the pace of the re-simulation data. For the front car distance, the state is populated with the equations below, assuming constant velocities within 100 ms trajectory sampling points and the maximum deceleration capability of $3m/s^2$ for ACC:

$$
\begin{aligned}
&d\ is\ the\ distance\ to\ the\ front\ car,\ as\ reported\ by\ perception,\\
&Vego\ is\ Ego\ car\ velocity,\ Aego\ is\ 3\ m/s^2\\
&Dsafe = 2 \times Vego\ is\ the\ safe\ distance\\
&if\ d > Dsafe \rightarrow "safe\_distance"\ state\\
&if\ d \leq Dsafe\ and\ d \geq Dsafe - 6 \rightarrow "acc\_distance"\ state\\
&if\ d < Dsafe - 6 \rightarrow "emergency\_distance"\ state
\end{aligned}
$$

Finally, the rule expresses three actions to be fulfilled as shown in table 2.

With these rules elements properly broken down, the following formal description can be constructed to represent the safety rule *goals*. The formal code looks like this:

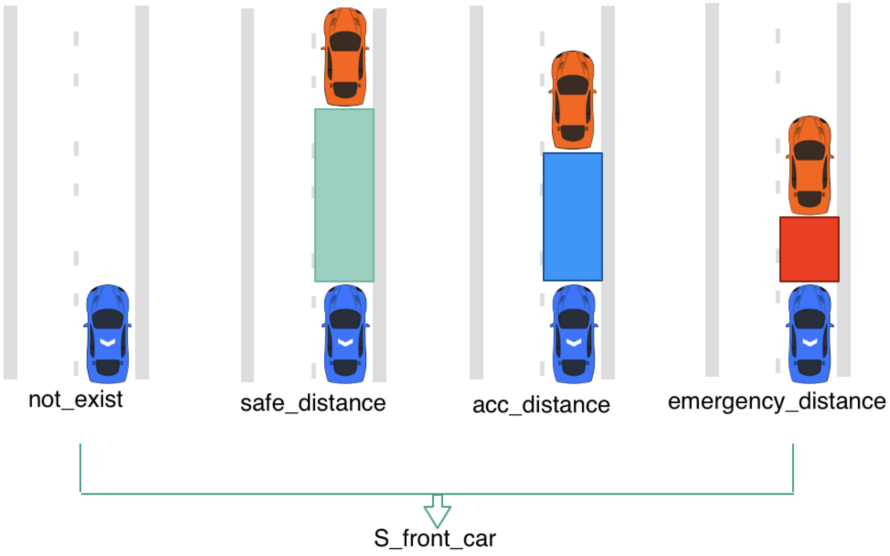

Fig. 5. Example of a state variable used to discretize the different safety distances taken into account between ego car and a front car. Depending on the current sensor readings and the assumptions used in the system, the front car distance can be classified as **not_exist**, meaning there is no visible front car, **safe_distance** meaning that considering current distance, velocities and assumptions for the front car accelerations there is no imminent risk of collision, **acc_distance** means that given the current assumptions and sensor readings, the ego-car would need to reduce its velocity at the ACC rate (3.5 m/s2) to satisfy the 2-seconds distance rule, while the **emergency_distance** value indicates that given the assumptions and current sensor readings, the ego-car would need to reduce its velocity at a higher rate than (3.5 m/s2) to avoid a collision.

Table 2. Actions to be fulfilled according to the longitudinal rule.

Type	Meaning
A_not_brake	No action
A_brake_acc	ACC braking ($0.6m/s^2$ to $3m/s^2$
A_brake_strong	$10m/s^2$ braking

when

S_traffic_jam **is** *yes*

then

goal type: constraint

when *S_front_car* **is** *acc_ distance* **then goal: executing** *A_ brake_ acc*

when *S_front_car* **is** *emergency_ distance* **then goal: executing** *A_ brake_ strong*

At this point, it must be noted that the formal constructs provided for Ego and front car are also applicable to the rear car (i.e. if a car follows). So, effectively, we have two sets of concurrent statements like the last 2 in the pseudo code above. This would be flagged, though, as infeasible by the language solver as priorities need to be added. Indeed, a priority is linked to responsibility levels according to the driving code: Ego car can only be held responsible for hitting the front car via a longitudinal maneuver. Hence, the statements above (related to the front car) must be indicated as having priority over the ones related to the rear car and this is done by changing the goal type statement to **goal type: priority** for the corresponding statements block.

Another aspect that we highlighted in section 2 is related to perception imperfections. It consists in the potential loss of track of objects over time. With sensors used in cars (cameras, radars, ultra-sound) and existing information fusion technology, the driving assistance system is subject to loss in tracking of objects due to inaccuracies in location and trajectory parameters estimations, as time passes. This results in objects IDs to disappear and new ones to be re-generated, potentially for the same objects.The safety rules have to deal with such case in order to decide for the validity of issuing an emergency maneuver action, for instance. Here, we talk about discretizing in time, collecting, via monitors, the disappearing times statistics from the re-simulation data. The corresponding "tracking" state values are then expressed with a statement like:

goal type: constraint
when *S_front_car_tracking* **is** *disappeared_ more_than_ t1*
then goal: executing *A_ emergency_ operation1*

In the statement above, "t1" is a statistical time value that needs to be evaluated by a monitor from re-simulation data described in this section. The *disappeared_more_than_t1* state is a Boolean created by a comparison with the threshold "t1" in the corresponding monitor.

The whole set of L3 rules, coded with the approach described above, generated a total number of 13500 checking states. These checks were formally compiled, without human intervention, into an executable checker code that was embedded into the Safety Assessment Tool introduced in section 3. This tool provided a global view over several hours of driving under traffic jam or dense traffic conditions, under day light or at night conditions. All safety violations as defined by the formal representation of the rules, i.e. triggering an action as presented above, were reported into a single view, along the timeline. The picture in Fig. 6 shows a graphical representation of an example of such report for a case of ACC braking that was reported as insufficient. The indicators on the left show the Ego car kinematic parameters (speed, acceleration, longitudinal, lateral). The situation is showing a merge to the left, into Ego car lane, of object labeled 14882 (zoom on the upper left), but the lanes structure is not reported yet in this representation. Object 14882 motion intent is depicted, at current time, by its kinematic projection trajectory model shown by a yellow color, in

Fig. 6. This trajectory model takes a statistical representation of the longitudinal and lateral speeds evolution over time from the current object position, based on a combination of its current acceleration parameter as well as worst case (strong) deceleration. The Ego car future trajectory is depicted in front of it and consists of 50 points separated by 100 ms. It is colored blue for the points that do not report safety issues and red for points that do. In the case shown below, the safety action is an A_brake_acc action that is required when the trajectory becomes red. For this situation, the insufficient braking level is reported due to the 2-second safe distance definition in the rule that is violated by car 14882 sudden arrival in Ego car lane. This was not anticipated by the motion planner during the road drive. In the functional system shown in Fig. 3, this safety error would be reported to the trajectory validation as a warning of a potential future issue. As we move over time in the Safety Assessment tool we can, hence, deduce whether that situation becomes real when the red color reaches the Ego car position (instantaneous violation), potentially highlighting a critical situation for which an emergency action is required. In the data set captured for this case, the instantaneous violation appeared roughly a second later.

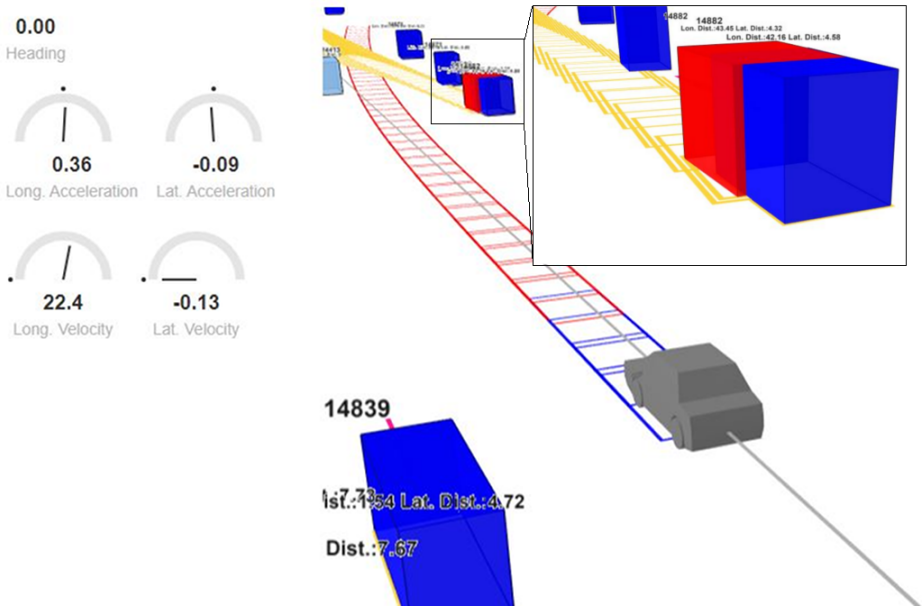

Fig. 6. Graphical representation of an ACC braking violation along Ego trajectory.

4.1 Notes regarding real-time performance

For the study presented in this paper, we did not only want to address the formal construction of car motion safety rules and their offline validation with re-simulation data captured on the road. We also wanted to make sure that the safety checker that is automatically built by the method studied was capable to be embedded within the car motion control software. And we wanted this to occur by using a real computing platform as used in the automotive industry. Such platforms are called ECUs in the automotive industry.

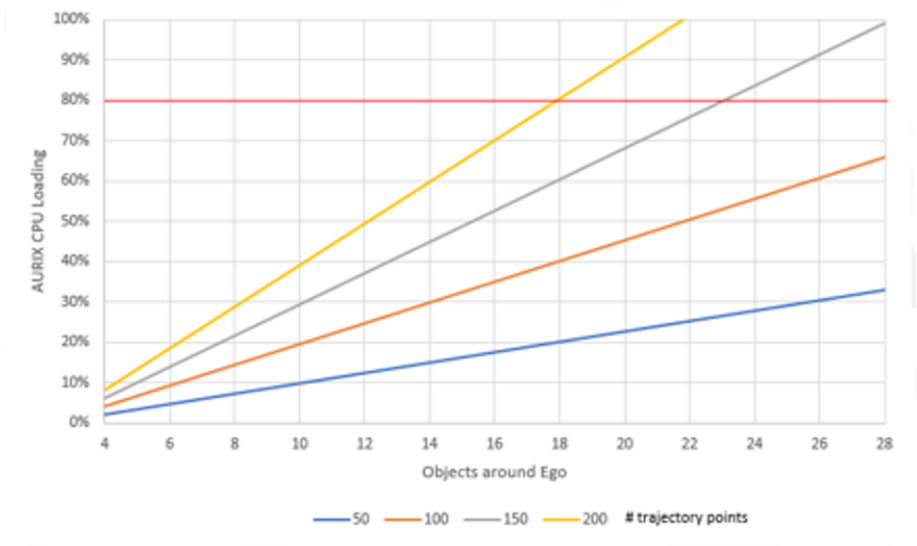

Fig. 7. Motion rules checker performance curves on a single AURIX core.

We chose a processing engine used in those and provided by the Infineon manufacturer [10], embedding a multi-core AURIX^{TM} system (in this study, a TCS397 development board), where we have constrained ourselves to operate on a single core to bound the processing footprint. The checker software code was generated as a single C++ code for both supporting safety assessment (offline) and for our performance analysis. The checker code was embedded into the IVEX Safety Co-Pilot runtime framework, which was responsible for updating the sensor data flowing into the monitors and for maintaining a memory and computationally efficient representation of the safety checker policy. The code was compiled with Infineon tools and run on the platform with re-simulation data patterns injected through the automotive bus ports available on that platform, at speed and synchronized (see table 1). The various situations found in the data, in terms of objects density around Ego car, and the capability to re-sample in time the trajectory allowed to gather curves like in Fig. 7. Our threshold

for considering that the checker is valid to run in a single care was that its execution did not exceed 80% loading at maximum frequency (300 MHz). These curves clearly indicate the fact that the checker can cope with a large number of surrounding objects for a given choice of trajectory sampling in the system.

5 Conclusion

This paper highlights real experiments, conducted over road captures made in the context of advanced driving and autonomous control car prototyping, on a practical approach to formalize the driving rules with an objective of maximal safety, using novel language and tools available from IVEX. Expressing safety rules applied to car motion control in a formal way carry challenges linked to both the imprecise nature of the rules defined by a human as well as the uncertainties related to the motion control process itself. We have presented, in previous sections, the needs for improvements from existing formal methods to address those challenges. And we have shown how a well thought set of language and tools, associated with a practical usage method, can handle all the above concerns together.

The method proposed also drives for a way of considering the safety rules verification in the chain that starts from motion planning and ends in car motion execution via physical actuators. Indeed, if it makes sense to apply safety principle within the various stages of the chain above, this paper has shown that the motion control safety rules generate a formal verification complexity of several thousand states. This shows that a safety checker executing this verification is required, associated with a trajectory validation function in order to cover the full safety complexity (see Fig. 3, above, for an example of this). So, this poses the question for the proposed approach of this paper to be compatible with real-time execution constraints of running within the electronic processing system in the car.

5.1 Next steps

The motion safety rules used for this study where a preliminary set. Our study allowed to show that, as a whole, they where performing as expected, capturing driving situations that where below the quality requirements. The analysis showed that the proposed method could be used to create indicators of bugs in safety rules coverage and system behavior. This part is worth further studying. Also, and finally, we have noticed that some rules where too "static" in their definition and that some parameters would benefit from being specified according to the driving situation (e.g., the safety distance of 2 seconds). This is another axis of future study.

References

1. Bezault, E., Howard, M., Kogtenkov, A., Meyer, B., Stapf, E.: Eiffel analysis, design and programming language. ECMA International, Tech. Rep. ECMA-367 (2005)
2. De Waen, J., Dinh, H.T., Cruz Torres, M.H., Holvoet, T.: Scalable multirotor uav trajectory planning using mixed integer linear programming. In: 2017 European Conference on Mobile Robots (ECMR). pp. 1–6 (2017)
3. Dinh, H.T., Cruz Torres, M.H., Holvoet, T.: Dancing uavs: Using linear programming to model movement behavior with safety requirements. In: International Conference on Unmanned Aircraft Systems. pp. 326–335. IEEE (2017). https://doi.org/https://doi.org/10.1109/ICUAS.2017.7991352, https://lirias.kuleuven.be/1571693
4. Dinh, H.T., Cruz Torres, M.H., Holvoet, T.: Combining planning and model checking to get guarantees on the behavior of safety-critical uav systems. In: ICAPS Workshop on Planning and Robotics. ICAPS Workshop on Planning and Robotics (2018)
5. Euro, N.: Euro ncap 2025 roadmap: In pursuit of vision zero. Leuven, Belgium (2017)
6. European Commission: Revision of the EU General Safety Regulation and Pedestrian Safety Regulation (2018), https://www.unece.org/fileadmin/DAM/trans/doc/2018/wp29grsp/GRSP-63-31e.pdf
7. Fisher, M., Dennis, L., Webster, M.: Verifying autonomous systems. Communications of the ACM **56**(9), 84–93 (2013)
8. Giacalone, J., Bourgeois, L., Ancora, A.: Challenges in aggregation of heterogeneous sensors for autonomous driving systems. In: 2019 IEEE Sensors Applications Symposium (SAS). pp. 1–5 (2019)
9. Gu, R., Marinescu, R., Seceleanu, C., Lundqvist, K.: Towards a two-layer framework for verifying autonomous vehicles. In: Badger, J.M., Rozier, K.Y. (eds.) NASA Formal Methods. pp. 186–203. Springer International Publishing, Cham (2019)
10. Infineon: AURIX^{TM} 32-bit microcontrollers for automotive and industrial applications (2020), https://www.unece.org/fileadmin/DAM/trans/doc/2018/wp29grsp/GRSP-63-31e.pdf
11. Maoz, S., Ringert, J.O.: Gr (1) synthesis for ltl specification patterns. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. pp. 96–106 (2015)
12. Microsoft: AURIX^{TM} 32-bit microcontrollers for automotive and industrial applications (2004), http://research.microsoft.com/en-us/projects/specsharp
13. Pek, C., Koschi, M., Althoff, M.: An online verification framework for motion planning of self-driving vehicles with safety guarantees. In: AAET-Automatisiertes und vernetztes Fahren (2019)
14. SAE International: Automated Driving Levels of Driving Automation are Defined in New SAE International Standard J3016 (2014), http://www.sae.org/autodrive
15. Schwarting, W., Alonso-Mora, J., Rus, D.: Planning and decision-making for autonomous vehicles. Annual Review of Control, Robotics, and Autonomous Systems **1**(1), 187–210 (2018). https://doi.org/10.1146/annurev-control-060117-105157, https://doi.org/10.1146/annurev-control-060117-105157
16. Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a formal model of safe and scalable self-driving cars. CoRR **abs/1708.06374** (2017), http://arxiv.org/abs/1708.06374

17. Wolff, E.M., Murray, R.M.: Optimal control of nonlinear systems with temporal logic specifications. In: Robotics Research, pp. 21–37. Springer (2016)
18. Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., Murray, R.M.: Tulip: a software toolbox for receding horizon temporal logic planning. In: Proceedings of the 14th international conference on Hybrid systems: computation and control. pp. 313–314 (2011)