Proc. of the European Safety and Reliability Conference (ESREL 2026)
Edited by José C. Matos, Paulo B. Lourenço, Michael Beer, Daniel Oliveira,
Edoardo Patelli, Hélder S. Sousa, Rui A. Silva and Monica Santamaria
 Published by Research Publishing, Singapore.
doi: 10.3850/ESREL2026061419_esrel26-p26219-cd

Hybrid Safety Verification for Autonomous Vehicles: Integrating CTL Model Checking with Continuous Safety Scoring

Shrijal Pradhan
German Aerospace Center (DLR), Germany. E-mail: shrijal.pradhan@dlr.de

Andrew Koerner
German Aerospace Center (DLR), Germany. E-mail: andrew.koerner@dlr.de

Björn Bahn
German Aerospace Center (DLR), Germany. E-mail: bjoern.bahn@dlr.de

When evaluating new algorithms for automated driving systems, human-comprehensibility is essential for comparative assessment. This enable researchers to benchmark iterative implementations against themselves and competing approaches across key metrics such as safety, efficiency, effectiveness, and performance. Simply comprehensible assessment is crucial in automated driving where researchers from many different fields bring their expertise.
Traditionally, tests are conducted with a binary verification approach. When conducting multiple validations, a formal model checking method such as a Computational Tree Logic (CTL) can be used. Such a method can verify the safety properties of a system but provides no input regarding the degree of success or the degree of failure. Such an information would be vital when considering an iteratively updating system.
This work proposes a quantitative grading system called the safety score that provides a human-comprehensible score for automated driving tests based on safety standards. The score can be viewed as a simple normalized score or a continuous grade in a known academic scale for intuitive human comprehension. This allows for the score to be easily used and interpreted in the field of automated driving where researchers from many fields work together for functional and system validation.
The score is implemented as an extension to the CTL model to provide a hybrid model checking approach. This hybrid approach enables stakeholders to assess both absolute safety guarantees through CTL verification (pass/fail) and relative performance quality through quantitative safety grades. This combination supports comparative analysis between systems, facilitates regulatory communication, and provides actionable feedback for iterative development. The hybrid model has been validated in both simulation and real-world scenarios on an automated research vehicle, demonstrating its capability to detect safety violations, quantify behavioral metrics, and provide actionable feedback via the open-source Eclipse ADORe (Automated Driving Open Research) framework.

Keywords: Autonomous vehicles, formal verification, model checking, runtime verification, safety scoring, safety assessment, ROS2, CTL, temporal logic.

1. Introduction

In recent decades, Automated Driving Systems (ADS) has seen drastic improvement in functionality and capabilities. This achievement has been obtained through several iterations of development and update through the collaboration of many different fields of research. These systems still need to go through many more tests and iterations if full autonomy is to be achieved (Benedikt et al. (2024)). In such a situation, it is important to have a testing platform where the test results can be easily interpreted by experts from different fields (Chen et al. (2021)).

Available testing platforms validate an ADS but do not provide easily interpretable feedback. These testing approaches provides a binary evaluation which does not contribute to understanding how a system can be iteratively updated (Gómez-Huélamo et al. (2022)). Such platforms are specifically difficult to manage for experts from different fields who might not specialize in Automated Driving (AD) testing (Maarssoe et al. (2025)).

The challenge for this work is to devise a novel hybrid verification framework that combines a

formal model checking framework that conduct absolute binary verification with a human comprehensible grading system that provides a score to assess relative quality.

In Section 2, the related literature is reviewed. In Section 3, the hybrid approach is introduced. Section 4, details the tests setup implemented and the results are presented in Section 5. In Section 6, the design choices of the method are discussed and concluded in Section 7. The test platform is open-sourced at (`https://github.com/DLR-TS/adore_model_checker`).

2. Related Work

In this section, previous works on relevant topics would be discussed.

2.1. *Model Checking*

Baier et al. (2016) explains that CTL and Linear Temporal Logic (LTL) are established temporal logics for of transition systems verification. CTL models timing based processes as a tree where each state can have multiple successor states, which can be used to map all acceptable future states. CTL has been successfully applied to Robot Operating System (ROS) based systems, particularly for verifying the Goal-Feedback-Result pattern in robotic applications by Wei et al. (2019). For these reasons it was selected as a base model checking for our hybrid approach.

2.2. *Scenario-Based Testing*

Steckhan et al. (2022) explains how scenario-based testing has emerged as a efficient way to validate automated driving functions. This method provides guidelines for virtual testing based validation that reduces time and testing effort by focusing on the most important testing situations. When integrating different functional modules from different technical teams, it becomes important to test for compatibility of the modified modules. Then the collective system can be tested for functional verification.

Neurohr and Möhlmann (2023) provides three advantages of virtual testing with scenarios. First, it provides a clearly defined goals regarding what the system should be able to collectively achieve. This single goal helps different teams collectively synchronize on a conceptual level during development. Second, individuals teams can make individual updates as testing locally what effects the new module has on the entire system. Finally, the scenario can then be migrated from simulation to real-world with reduced effort.

Huber et al. (2020) evaluates different driving characteristics and the respective evaluation metrics. Criticality metrics are used to measure the behavioral validity of the driving functions in terms of traffic conflicts. These metrics focus on the evaluation of behavioral validity and not on functional validity of the driving system.

2.3. *Safety Metric*

Wishart et al. (2020) provides a more comprehensive evaluation of the safety performance metrics. This work includes criticality metrics for behavioral analysis and driving performance metrics that are more suited for functional testing. The focus is towards the evaluation of the metric itself than towards emprical anaylsis.

Volk et al. (2020) proposes a method to evaluate safety of the perception system in automated driving. The formalized safety metric uses threshold values to normalize standard perception metrics. The incorporation of parameters of different ranges into a single assessment score improves the interpretability of the evaluation. The assessment score is then classified into a pre-defined set of classifications that helps with quick evaluation of a test scenario. These methods allow for an effective comparison of the performance of different systems. The focus of this work is on perception tests and not on the testing of driving functions.

3. Methodology

In this section, the workings of the hybrid model checking approach is explained.

3.1. *CTL Model Checking*

Formal verification of autonomous vehicles in variable and complex environments remains a significant challenge due to the computational complexity and the need for real-time decision making. We adopt CTL as the temporal logic founda-

tion primarily to establish a baseline approach for testing they hybrid approach.

CTL's branching-time semantics allow us to reason about multiple possible execution paths from any given state, which aligns well with the non-deterministic nature of autonomous driving scenarios where multiple valid behaviors may exist at decision points. CTL provides sufficient expressiveness for our proof-of-concept while maintaining computational tractability.

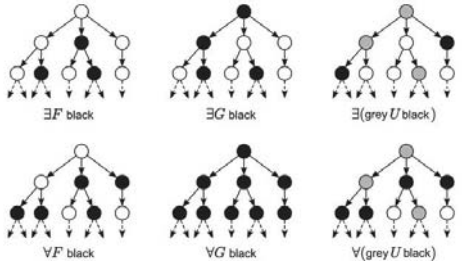

Fig. 1.: CTL Graph from Thomas et al. (2022).

3.2. *Safety Scoring Mechanism*

Safety score takes a feature of the driving behavior as input and produces a score regarding the similarity of the input to expected behavior. These factors depend on the type of scenario and the behavioral feature being tested. Testing different features individually allows for the sensitivity of the features to be set based on the scenarios by setting the appropriate parameters.

$S \in$	Classification	Status
[0.0–0.2]	**insufficient**, high fatality risk	Fail
(0.2–0.4]	**bad**, serious violation risk	Fail
(0.4–0.6]	**good**, noticeable violation	Pass
(0.6–0.8]	**very good**, moderate violation	Pass
(0.8–1.0]	**excellent**, high safety prob.	Pass

Table 1.: Safety classification based on violations.

Safety score bounds input data into acceptable values from previous tests and the rules of the road the driving feature has to follow. For driving behaviors where a certain specified empirical value is expected such as speed limit compliance, safety score can be inversely associated with the value by which the speed limit is exceeded. In such a situation, driving without violations would interpret to a perfect score.

3.2.1. *Classifying Safety*

The safety classification table from Volk et al. (2020) is updated to Table 1 to associate better with driving violations. To make it easier for researchers and engineers from different fields to test their systems, a grading system is also provided for the safety score based on american and german schooling standards in Table 2.

The classification of safety from the score above 0.4 to 1.0 generally indicates an acceptable driving situation. A score from 0.0 to 0.4 is an indication that the driving behavior is not acceptable based on the set parameters and the system should fail. A classification of excellent indicates minimum violations and optimal driving conditions. A very good classification indicates some violations that are generally accepted conditions for driving. The class good indicates a gray zone which borders noticeable violations and acceptable driving conditions; the system has not failed yet but it is recommended to improve it. The class bad is an indication that the violations were critical and the system fails the test. The end class of insufficient is available to get a measure of the margin by which the system has failed and indicates high risk if the system is operational.

$S \in$	Class.	DE	US
[0.00–0.40]	**insuf., bad**	5.0	F
(0.40–0.45]	**good**	4.0	D-
(0.45–0.50]	**good**	4.0	D
(0.50–0.55]	**good**	3.7	D+
(0.55–0.60]	**good**	3.3	C-
(0.60–0.65]	**very good**	3.0	C
(0.65–0.70]	**very good**	2.7	C+
(0.70–0.80]	**very good**	2.3	B-
(0.80–0.85]	**excellent**	2.0	B
(0.85–0.90]	**excellent**	1.7	B+
(0.90–0.95]	**excellent**	1.3	A-
(0.95–1.00]	**excellent**	1.0	A

Table 2.: Safety grading (DE=German, US=American).

3.2.2. *Bounding Conditions*

For a feature selected for safety scoring, the acceptable input values are first obtained through rigorous testing. These acceptable values are then bound to the safety score within the range of 0.0 to 1.0 where latter represents guaranteed safety. The optimal value is bound to score of 1.0. The least acceptable value that is associated with the maximum allowed unwanted deviation from the optimal score is bound to the value of 0.2. The score from 0.2 to 0.0 is generally reserved for safety based research and is not directly used for empirical testing. The remaining bounds of the classification as presented in Table 1 are assigned to input values within the acceptable range based on analysis from testing.

When a new input data is received, the value is associated with a classification. It is then normalized and scaled to obtain a safety score.

Signed Normalization Function normalizes inputs within the bounds of the specified classification. The lower bound, l; is associated with lower safety score and the upper bound, u; is associated with higher safety score. This helps to create a directly proportional relation with the safety score regardless if the input data is directly or inversely proportional to the safety score as

$$f_{sn_{l,u}}(x) = \frac{x - l}{u - l}. \tag{1}$$

Scaling Function is used to scale the normalized value within the bounds of the classification to obtain a safety score. In this case, the upper bound safety, u; is higher than the lower bound safety, l such that

$$f_{scale_{l,u}}(x) = (u - l) * x + l. \tag{2}$$

3.2.3. *Guarding Conditions*

Guarding conditions allow for defining specialized conditions that have to be held true for the system to pass. These conditions might change based on the scenario or the characteristic that is evaluated. This provides an additional dimensional to the tests. If the guard conditions do not hold, the safety score is over-written by the guards to signify a fail. For each individual test a set of guard conditions can be implemented. This exclusivity allows for different conditions to be set for different behaviors.

Guard Function is used to overwrite the safety score, s; with a pre-defined score, y_i; if conditions, z_i; does not hold for input x. The function takes a vector of conditions, $\mathcal{Z}$; and a vector of pre-defined scores, $\mathcal{Y}$; as input. Each row in the pre-defined vector is associated with the respective row in the conditions vector such that

$$f_{gd_{\mathcal{Y},\mathcal{Z}}}(x, s) = \begin{cases} s & \text{if all } z_i(x) \in \mathcal{Z} \text{ satisfied ,} \\ y_i & \text{if } z_i(x) \in \mathcal{Z} \text{ not satisfied.} \end{cases} \tag{3}$$

The pre-defined scores vector is hierarchical such that the lowest score is at the top. If multiple conditions from the vector $\mathcal{Z}$ are not satisfied, the hierarchy must be followed to output the least possible safety score.

3.2.4. *Run time Safety Scoring*

When evaluating a run time system, a time interval is defined to extract scenes from a scenario. For each scene, a safety score is assigned to the characteristic evaluated. An average safety score is calculated based on the current scene and previous scenes. The final safety score is calculated at the end of the tested scenario. This setup tolerates minor violations in the behavior as long as the overall score is acceptable. With the run time evaluation, it is possible to analyse the current past inputs to obtain a measured safety score.

3.3. *Hybrid Verification Approach*

To understand the hybrid method, consider the model in Section 3.1 as a collection of blocks of tests. For example, the speed tests would make one block and the lane tests another block, each with it's own logic tree. All the blocks have to pass for the complete model checker to give a pass statement. A binary output block can be replaced with the safety score block that takes input in the form of data from current and previous time-steps for a specified scenario. This data is processed based on the specified bounding conditions, as explained in Section 3.2.2, and the set guard conditions, as explained in Section 3.2.2. The output is a score

based on Section 3.2.1 that can be converts to a binary value by referencing Table 1.

4. Implementation

The model checker is implemented in Python 3.8 and supports the ROS2 versions of Foxy, Galactic, Humble and Iron. It has two operating modes. The model checker runs in online monitoring mode to check a ROS instance. The offline analysis mode is used to check the propositions on bag files.

A yaml file configures the data source mapping, enables safety proposition groups, sets safety parameters, thresholds, monitors frequency, buffer settings and an evaluation function that supports the CTL expressions from Section 3.1.

For testing we integrated the hybrid model checker into the ADORe framework. ADORe is a modular autonomous driving research platform that provides a comprehensive software architecture for developing and evaluating autonomous vehicle systems. The framework was deployed to DLR's research vehicle, which is equipped with state-of-the-art sensors including LIDAR for precise distance measurements, GNSS for accurate positioning, and multiple cameras for environmental perception (Maarssoe et al. (2025)), as shown in Figure 4. This sensor suite enables real-world data collection and validation of the model checking approach under diverse driving conditions. For testing we implemented a dedicated user interface that visualizes and summarizes the model checking reports in a human-comprehensible format, providing intuitive feedback on safety property verification results as demonstrated in Figure 2. The interface allows researchers to quickly assess the verification status and identify potential safety violations during autonomous driving scenarios. Our test vehicle ViewCar2 is a VW Passat Variant with 4.88 meters in length and 1.83 meters in width.

The implementation is done for two blocks, the lane compliance block and the speed compliance block. As explained in Section 3.3, for each block, the CTL based binary testing containing a single pass criterion is replaced by the safety scoring mechanism's multiple criteria and provide an accumulated score over time.

The penalization is divided into six criteria, four deviation criteria for the bounding conditions from Section 3.2.2 and two guard criteria from Section 3.2.3 for repeated violations. General expected tolerance, lower tolerance, upper tolerance and exceeded tolerance make up the four deviation criteria. General expected tolerances are what is seen with general driving behavior. They are not safety critical given other safety factors do not show any warnings. Repeated violation of an infringement however, might indicate functional issues with the system. The two guard criteria are devised to penalize the system more strictly for exceeding a set number of infringements.

The safety scores are bounded to the respective tolerance criteria. The lower range values of the criteria are bound to a score slightly lower than lower bounds of a selected safety classifications from Table 1 that it closely associates with. In the run time systems, this means that the system would be consistently penalized for repeated infringements and would fall to a lower safety class if the infringements persist. Important to note here is that a single infringement does not equate to rigid penalty for the system.

The Equations 1 and 2 are combined with run time capabilities to obtain the safety score. To calculate the safety score, the input data is first bound to an absolute upper and an absolute lower safety bound represented by the deviation criteria. Beyond the absolute upper bound the system is always safe and beyond the absolute lower bound the system is always unsafe. The input is normalized with the deviation criteria bounds that it falls under and then scaled with the safety scores associated with the deviation criteria. Given a deviation criteria with lower bound deviation, l_d; and upper bound deviation, u_d; that are associated with lower safety bound, l_s; and upper safety bound, u_s; for a time, $t_i \in \mathcal{T}$; the safety score is

$$S_{t_i,l\_d,u\_d,l\_s,u\_s}(x) = f_{scale_{l\_s,u\_s}}(f_{sn_{l\_d,u\_d}}(x)). \tag{4}$$

In a run time system the safety score can be calculated based on observations from the current time step and the previous time steps. For a vector of time $\mathcal{T}$ with steps from some arbitrary previous

time to the current time, $t_f \in \mathcal{T}$; safety score can be represented with all $t_i \in \mathcal{T}$ such that

$$S_{\mathcal{T}} = [S_{t_0}, ..., S_{t_f}]^T. \quad (5)$$

The agglomerated safety score for current time, $t_f \in \mathcal{T}$; is then an average of all the safety scores in $S_{\mathcal{T}}$ such that

$$S_{\mathcal{T}_f} = \overline{S_{\mathcal{T}}}. \quad (6)$$

A guarded safety score can then be formalized with Equation 3 for the current time, $t_f \in \mathcal{T}$; with vector of all inputs till current time, X_f; as

$$S_{gd_{\mathcal{T}_f,\mathcal{Y},\mathcal{Z}}}(X_f, S_{\mathcal{T}_f}) = f_{gd_{\mathcal{Y},\mathcal{Z}}}(X_f, S_{\mathcal{T}_f}). \quad (7)$$

4.1. *Lane Keeping*

To ensure the vehicle stays on the road, a lane keeping test is implemented to maneuver safely within the lane. The system is penalized for infringing the lane boundaries.

Description	Dev. (m)	Score	Guards
Expected tol.	[0.0, 0.3)	[1.0, 0.7)	Min 50%
Lower tol.	[0.3, 0.5)	[0.7, 0.35)	–
Upper tol.	[0.5, 0.7)	[0.35, 0.2)	–
Exceed tol.	0.7+	0.2	Max 3

Table 3.: Lane keeping tolerance and thresholds (Dev.=Deviation from centerline).

A generic road on the DLR campus in Brunswick is chosen for the test map where the lane is $3.5m$ wide. The tolerance values given in Table 3 are based on this road, the dimensions of the vehicle and fine-tuning from test data. The ranges in the deviation column are the possible inputs to the lower bound, l_d; and upper bound deviations, u_d; to complete Equation 4. It should be noted that the deviation and the safety score have an inverse relationship, from each column the higher deviation from the range relates to the lower bound deviation. If the upper lane tolerance (more than $0.7m$ from center line of the lane) is exceeded more than 3 times during the test, it fails. It also fails if more than 50% of the trajectory is outside of the smaller expected corridor (max. $0.3m$ deviation from center line of the lane). They form the guard conditions vector, $\mathcal{Z}$; associated with a scores vector, $\mathcal{Y}$; where all values are 0.2. These variables are inputs to Equation 7 that formalize the lane keeping tests. The guard conditions and the agglomerated bounding conditions are evaluated at the end of the scenario after data is collected for the entire drive.

4.2. *Speed Excess*

The test road on the DLR campus is set with a speed limit of $30km/h$ or $8.33m/s$. The tolerance values given in Table 4 are based on this limit and fine-tuning from test data. Compared to the lane deviations, it can be seen that the upper tolerance infringement is more strictly penalized due to the fact that the deviation associated accounts for significant increase in speed. As with Section 4.1 the values from the table are used as input into Equation 7 to formalize the speed tests. The lower tolerance is set at maximum $2m/s$ more than the speed limit with the upper tolerance at $4m/s$. If the upper tolerance is exceeded more than 5 times the test will fail as well as if the lower tolerance is exceeded more than 50%.

Description	Speed Dev. (m/s)	Score	Guards
Expected tol.	[0.0, 1.0)	[1.0, 0.6)	Min 50%
Lower tol.	[1.0, 2.0)	[0.6, 0.3)	–
Upper tol.	[2.0, 4.0)	[0.3, 0.15)	–
Exceed tol.	4.0+	0	Max 5

Table 4.: Speed excess tolerances and thresholds (Dev.=Deviation from speed limit).

5. Experimental Results

Experiments are conducted for the lane and speed tests. To show the advantage of the hybrid method, the lane experiment show a test that barely and the speed experiment shows a fully successful test.

Figure 2 show a lane test that barely passes. In the test, the vehicle partially infringes to upper tolerance without triggering the guard conditions presented in Section 4.1. With the CTL logic, any infringement to a set tolerance is considered a test failure.

Fig. 2.: Lane Report.

However, with the hybrid method the test performance can be gradually penalized based on minor, medium or major infringement. The system would then not fail if the tolerance is violated for a small number of times, which is a more natural method for driving functions. Further, in the test, the hybrid method provides a barely pass grade of "D" from Table 2. It can be concluded that the lane keeping system does pass the test based on the tolerance criteria and obtains a "good" safety classification from Table 1 but can be significantly improved.

Figure 3 shows the successful speed test where the vehicle maintains a speed of $8.33m$ or less as explained in Section 4.2. This test would pass with a CTL logic and with the hybrid method presents a grade of "A" from Table 2.

Fig. 3.: Speed Report.

6. Discussion

We selected CTL as our temporal logic formalism primarily due to its well-established theoretical foundation and straightforward implementation. CTL provides sufficient expressiveness to implement our hybrid approach, enabling the specification of safety properties relevant to autonomous driving scenarios while maintaining computational tractability.

Fig. 4.: DLR Research Vehicle.

The hybrid approach presented in this work addresses a critical gap in autonomous vehicle evaluation by bridging the divide between formal verification and practical assessment needs. Traditional model checking provides rigorous safety guarantees but offers limited insight into system performance quality, while purely quantitative metrics lack the formal assurance required for safety-critical applications. Our dual scoring system enables stakeholders to simultaneously assess absolute safety compliance through CTL verification and relative performance quality through intuitive academic grading scales. This combination supports the iterative development process common in autonomous vehicle research, where engineers need both formal validation of safety properties and actionable feedback for system improvement.

The integration with the ADORe framework and ROS ecosystem demonstrates the practical applicability of our approach within existing autonomous driving research workflows. By providing open-source availability the framework lowers the barrier to adoption for researchers who may lack extensive formal methods expertise. The real-world validation on DLR's research vehicle confirms that the approach can handle the complexity and uncertainty inherent in actual autonomous driving scenarios, making it a viable tool for both academic research and industrial development of autonomous vehicle systems.

7. Conclusion and Future Work

This approach has several acknowledged limitations that present opportunities for future work. While CTL supports branching-time reasoning, our current safety scoring implementation follows a linear-time assumption along single execution paths. Despite these constraints, the hybrid approach's modular design makes it extensible to other temporal logics, suggesting that future research could integrate LTL or probabilistic temporal logics to enhance expressiveness and better handle uncertainty in autonomous driving environments.

Acknowledgement

The authors would like to thank the ADORe development team for their contributions to the autonomous driving framework.

References

Baier, C., J.-P. Katoen, and K. G. Larsen (2016). *Principles of model checking*. MIT Press, The.

Benedikt, M., E. Böde, S. Burton, W. Damm, M. Fränzle, E. Möhlmann, and P. Rosenberger (2024). Controlling risk for highly automated transportation systems operating in complex open environments. SafeTRANS Closing the Gap Initiative.

Chen, J., S. Li, and M. Tomizuka (2021, 02). Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems PP*, 1–11.

Gómez-Huélamo, C., J. Del Egido, L. M. Bergasa, R. Barea, E. López-Guillén, F. Arango, J. Araluce, and J. López (2022, January). Train here, drive there: Ros based end-to-end autonomous-driving pipeline validation in carla simulator using the nhtsa typology. *Multimedia Tools Appl. 81*(3), 4213–4240.

Huber, W., A. Lauer, and R. Graubohm (2020). Scenario-based testing for automated driving functions. *ATZ Worldwide 122*, 56–61.

Maarssoe, M. S., S. Konthala, M. Mizdrak, G. Lucente, M. Nichting, T. Lobig, and A. Koerner (2025). Adore: Unified modular framework for vehicle and infrastructure-based system level automation. In *Proceedings of the 11th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS*, pp. 571–581. INSTICC: SciTePress.

Neurohr, B. and E. Möhlmann (2023, 02). Scenario-based verification and validation of automated transportation systems. *INSIGHT 25*, 47–50.

Steckhan, L., W. Spiessl, N. Quetschlich, and K. Bengler (2022). Beyond sae j3016: New design spaces for human-centered driving automation. Berlin, Heidelberg, pp. 416–434. Springer-Verlag.

Thomas, C., M. Cosme, C. Gaucherel, and F. Pommereau (2022, 06). Model-checking ecological state-transition graphs. *PLOS Computational Biology 18*, e1009657.

Volk, G., J. Müller, N. Navarro, M. Stählin, and B. Penzkofer (2020). Towards a safety argumentation for perception functions in automated driving. *IEEE Intelligent Vehicles Symposium*, 487–492.

Wei, W., X. Li, Y. Guan, R. Wang, Q. Lu, and J. Zhang (2019). Model checking for the goal-feedback-result pattern in ros. In *2019 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pp. 640–645. IEEE.

Wishart, J., S. Como, J. Elli, and S. Ghanipoor (2020). Driving safety performance assessment metrics for ads-equipped vehicles. *Transportation Research Record 2674*, 173–180.