THINK INDIA JOURNAL

ISSN: 0971-1260

Vol-22-Issue-17-September-2019

Comparative Analysis of Graph Coverage Criteria based Test Paths of Automotive embedded system for validation and safety

Parampreet Kaur, Rajeev Sobti
School of Computer Science Engineering
Lovely Professional University
Jalandhar, India
parampreet.18758@lpu.co.in , *sobtirajeev@gmail.com*

Abstract

The automotive modernizations in the past decades are dominated by new software-based functions. A validation of these functions is always required for ensuring the safety of the vehicles. Several studies show the effectiveness of design-based methodologies for performing testing and evaluation of vehicular system functionalities. Along with the advancement of automated driving technologies, the demand for simulation-based assistance of such vehicles is increasing. For the simulations, the modelling of the observation sensors plays an indispensable role. State-of-the-art studies and research investigations elucidates an ever-increasing requirement of efficiency of the test paths. The elementary structure of the modelling is also delineated here, and a viewpoint for future requirements on it is given. Prime path coverage and edge-pair coverage when combined with prefix graph algorithm generates least number of yet effective test cases which can traverse all the states in less time. Other approaches such as node coverage or edge pair coverage also yields good results but the test path requirements are very large.

Introduction

Advanced autonomous vehicles either being used for private transportation or freight delivery can offer a doubtlessly great disruption to lifestyles, commercial enterprises and society [1]. The possible benefits are manifold - reductions in accidents bobbing up from human mistakes, reduced fee & environmental impact of transport, liberation of time presently committed to driving, and accessibility to a wider variety of users are all theoretically addressable.
Keeping it into consideration, some key challenges should be conquered to reap this imaginative and perceptive aspect [2].

1. **Guarantee of quality structures and software programs**: How can we outline and show the proper degree of acceptability?
2. **Sensing and Connectivity**: How can we ensure the right relationship between an automobile and its surroundings?
3. **Judgement**: How can automated systems exercise judgement?
4. **Architectures for managing complexity**: How are we able to manage the resulting machine complexity?
5. **Verification & validation:** To what extent is testing required, and the how it can be achieved?

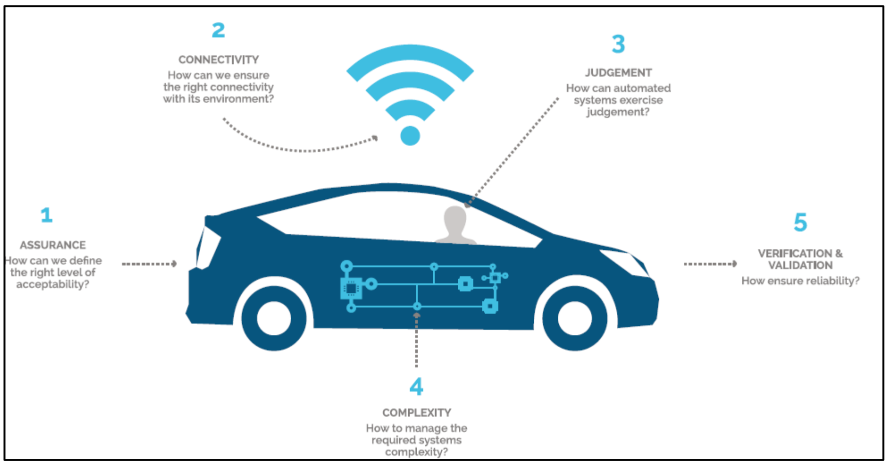
2
CONNECTIVITY
How can we ensure the right connectivity with its environment?
3
JUDGEMENT
How can automated systems exercise judgement?
1
ASSURANCE
How can we define the right level of acceptability?
5
VERIFICATION & VALIDATION
How ensure reliability?
4
COMPLEXITY
How to manage the required systems complexity?

Figure 1: Key Challenges in automated driving

The assessment and testing of autonomous vehicular systems is gaining widespread attention and multifaceted nature as the system's complexity increases [3]. To handle this effectively, the need for well-organized and widely accepted opportunities to verify these systems ascends. Physical road examinations would necessitate millions of kilometres in order to obtain the vehicle's operation. Therefore, this method of validation and verification is not appropriate for a development process, such as the V-model [4]. Automatic driving roles have to be assessed in a diversity of complex traffic situations, with the indecisions resulting from different accommodating traffic members. Computer-generated assessment by simulation capacitates to provide a harmless way to evaluate the performance of algorithms under a wide-ranging variation of scenario constraints. Simulation based strategies have proven to be very engaging and trustworthy means to conform the reliability and safety of the driving applications by virtually displaying the behaviour of the embedded system sensors and actuators. [5]. The Figure 3 provides overview of different test-based architectures used in automotive sector.

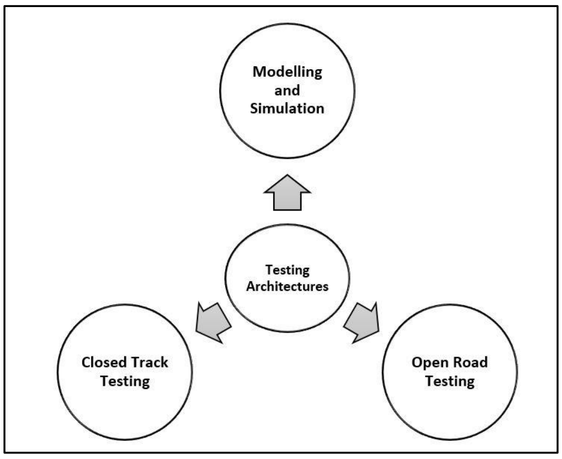

Figure 2: Testing Architectures

Related Work

Task-Specific self-directed cars system validation method is modelled in functional levels, evading the complexities of different algorithms estimation, comparable to Grey-Box testing [6], described in Figure 4. By analysing autonomous driving systems, lists of simple function test cases are designated and accumulated into different testing procedures, which are further abstracted as driving process classes. By analysing precise driving functions, autonomous driving features can be estimated. Self-driven functional tests are approved under different simulation or real environments. By a recognized testing method, such as test-designing, planning, logging and valuation and accomplishment confirmation, all vehicle operations are lastly assessed with diverse task complication property and dissimilar environment difficulties [7].

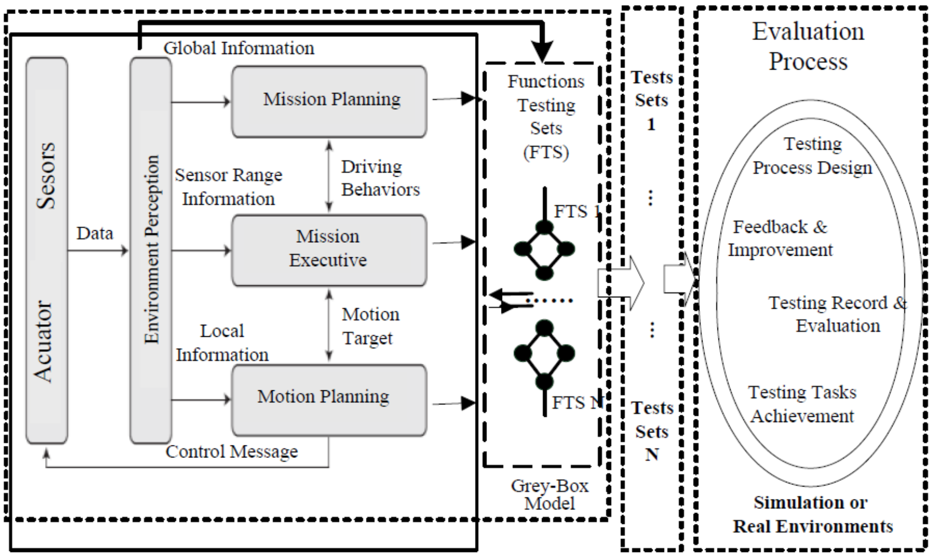

Figure 3: Vehicular System validation modelling

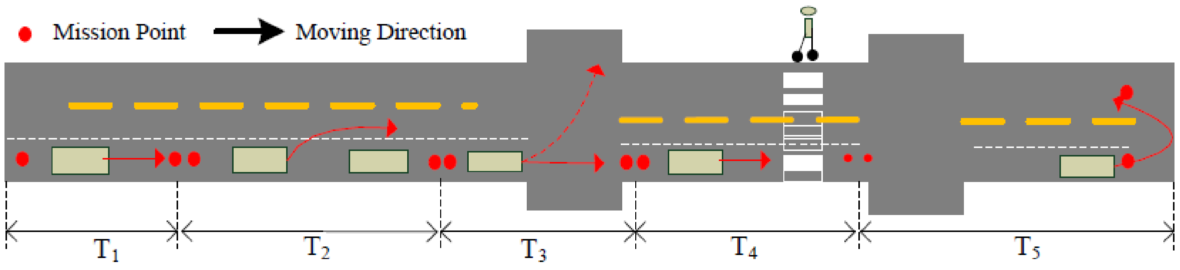

Figure 4: Different states of driving tasks

The self-governing vehicular growth boons with a description of the functional and behavioural requirements/specifications of the anticipated functions, from the elementary self-directed driving requirements to switch small and long term planning, escape unsafe crashes and accidents [8]. The autonomous vehicle validation approaches comprises of “Test Drive” and “VEHIL” [9] simulation techniques, and the collective evolutionary testing.

Model-based testing reintroduces the complete process of functional software testing: from commercial requirements to the testing databases, with manual or automated test implementation. It helps during the phases of designing and creating tests, recording the test data-repository, making and preserving the traceability matrices between test-paths and test-requirements, and fast-tracking testing automation. Testing is one of the most significant instruments to validate the accurateness of embedded systems [10]. A functional test perceives a failure if the observed and the specified behaviour of the SUT differ. Model-based testing is about using models as specifications. Numerous modelling languages have been useful to generate test models, the Unified Modelling Language (UML) [11], or the Object Constraint Language (OCL) [12]. Model-based testing involves deriving test suites automatically from formal test models. This paper is focused on automatic model-based test generation with UML state machines and OCL expressions [13].

This paper [14] targets at presenting on the notable coverage of EAST-ADL as an abstract model for the comprehensive scope of pertinent models in the automotive field, and secondly, the prominent level of integration of novel highly developed analyses, for safety and effectiveness in specific, and optimization competences to make satisfactory use of the throughput only powerful processors can offer. Authors presented the EAST-ADL procedure, explaining in detail timelines and the most useful abstraction stages. Fundamentals for model-based optimization are also obtainable as the chance of dealing with contradictory areas.

A Test path can be attained using the Depth First Search technique (DFS), by navigating an LTS beginning from the original state. The coverage criterion of the methodical process is that all paths conditions and labelled shifts are traversed, i.e., all states and labelled edges are visited at least once. Since we are deliberating on functional testing, whole coverage is a rational and achievable goal, to assure a systematically examination of the attribute functionalities [15].

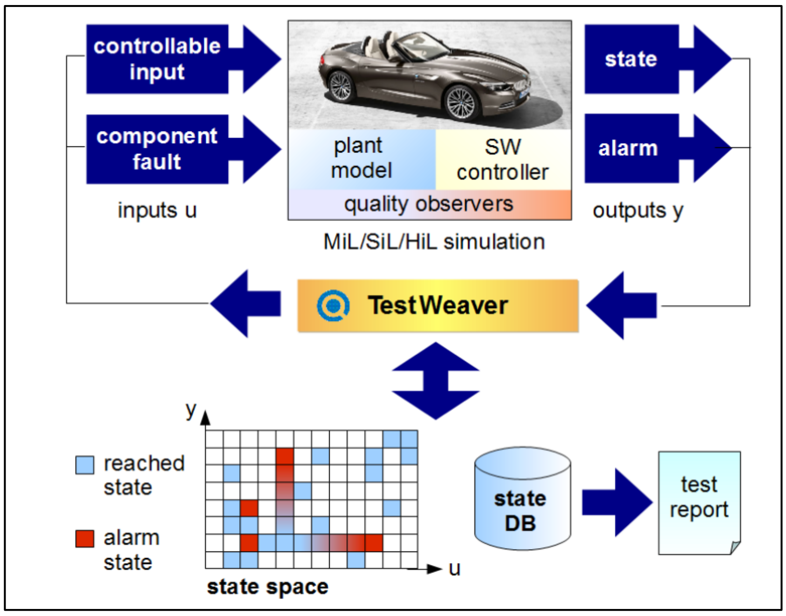

Figure 5: MIL/SIL model as described in [16]

Test-Weaver [16] controls definite constraints and inputs and it perceives accuracy / quality pointers, as well as other particular state variables curtailing from the controller or from the car and traffic virtual model. The networking between Test-Weaver and the virtual environment is done at distinct time points. At these time intervals snapshots of the contributions and of the observables are communicated to Test-Weaver.

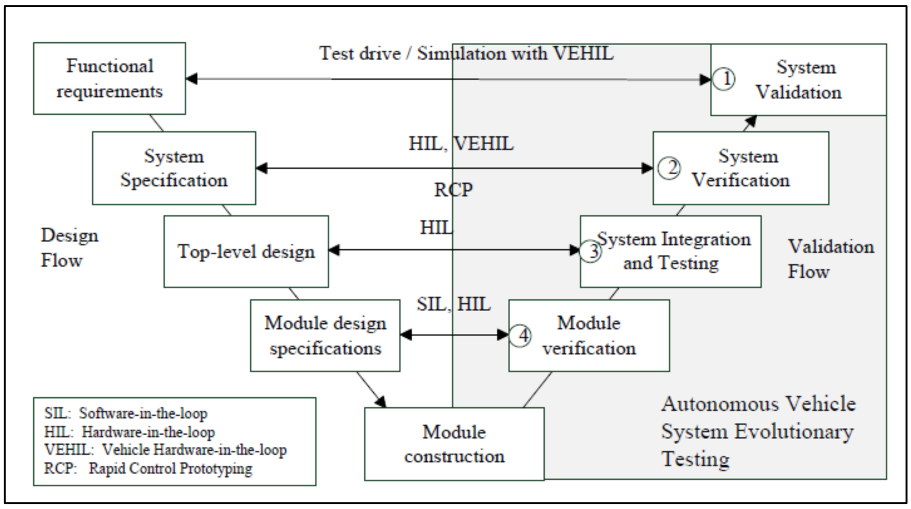
Functional requirements
Test drive / Simulation with VEHIL
System Validation
System Specification
HIL, VEHIL
System Verification
RCP
Design Flow
Top-level design
HIL
System Integration and Testing
Validation Flow
Module design specifications
SIL, HIL
Module verification
SIL: Software-in-the-loop
HIL: Hardware-in-the-loop
VEHIL: Vehicle Hardware-in-the-loop
RCP: Rapid Control Prototyping
Module construction
Autonomous Vehicle System Evolutionary Testing

Figure 6: Automated Vehicle evolutionary design and testing workflow diagram

Proposed Work

We have performed validation using various types of test coverage criteria. These include, Edge Coverage, Prime Path Coverage, Prime Path Coverage using the prefix graph algorithm, Edge-Pair Coverage, Edge-Pair Coverage using the prefix graph algorithm. The system states taken into consideration are Adaptive Cruise Control and Lane change and keep assist functions. The required number of states are transformed into a simplified finite state chart as described in the figure 8. By keeping into view the importance of modelling during every phase of software development, we have followed the Double V-model as described in the Figure below for drawing comparison between various graph-based coverage criteria methods.

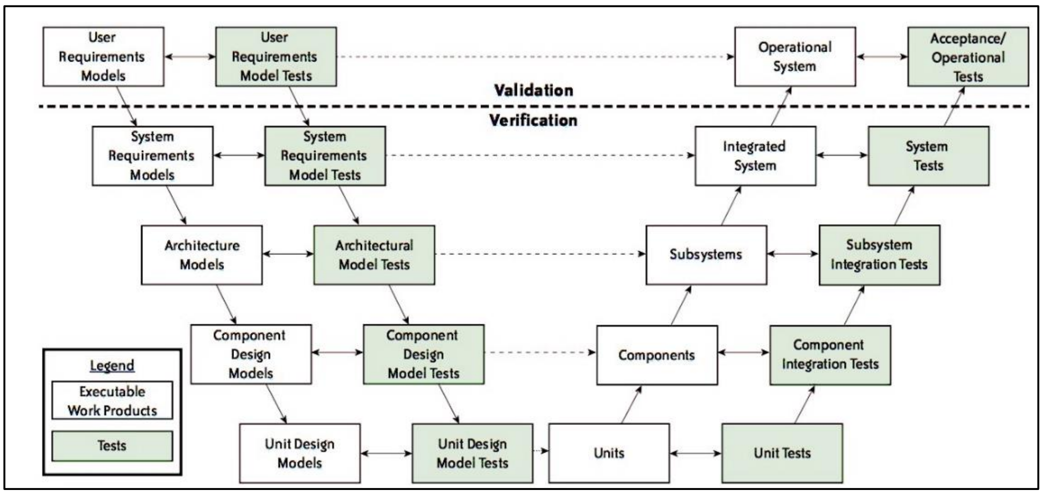

Figure 7: Double V-Model for system verification

The system functions are later converted into a simplified graph-based structure where first time analysis is done keeping only 1 initial node. Later more initial and final nodes are added to check the algorithm's capability to identify and execute test paths for verification.

ISSN: 0971-1260

Vol-22-Issue-17-September-2019

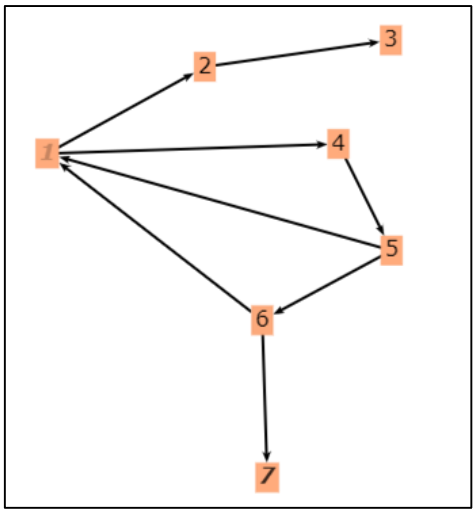

Figure 8: Finite state machine representation

The graph provided in the figure 8 above consists of 7 states which an automated vehicle is intended to reach. The test cases and test paths thus provided by each of the different algorithms will identify the best and optimized test requirements to traverse these nodes which are the states required to be reached. To execute the test path generation, we have used edge coverage, edge-pair coverage with and without prefix graph algorithm. Three test paths are needed for Edge Coverage as shown below in Table 1. Three testing-paths are needed for Prime-Path Coverage using the prefix-based graph algorithm as shown in the Table 2.

Table 1: Edge Coverage Method Test Cases/Requirements

[1,4,5,6,1,4,5,6,7]
[1,4,5,1,2,3]
[1,4,5,6,7]

Table 2: Prime Path Coverage Analysis of TEST CASES

Test Paths	Direct Test-Requirements
[1,4,5,1,4,5,1,2,3]	[4,5,1,2,3], [4,5,1,4], [1,4,5,1], [5,1,4,5]
[1,4,5,6,1,4,5,6,7]	[4,5,6,1,4], [1,4,5,6,7], [1,4,5,6,1], [6,1,4,5,6], [5,6,1,4,5]
[1,4,5,1,4,5,6,1,4,5,6,1,4,5,6,1,2,3]	[4,5,6,1,2,3], [4,5,6,1,4], [1,4,5,6,1], [6,1,4,5,6], [5,6,1,4,5], [4,5,1,4], [1,4,5,1], [5,1,4,5]

ISSN: 0971-1260

Vol-22-Issue-17-September-2019

Testing-Paths	In-Direct Test-Requirements
[1,4,5,1,4,5,1,2,3]	[4,5,1,2,3], [1,4,5,1]
[1,4,5,6,1,4,5,6,7]	[1,4,5,6,7]
[1,4,5,1,4,5,6,1,4,5,6,1,4,5,6,1,2,3]	[4,5,6,1,2,3], [4,5,6,1,4], [1,4,5,6,1], [6,1,4,5,6], [5,6,1,4,5], [4,5,1,4], [1,4,5,1], [5,1,4,5]

Table 3: 3 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Testing-Paths	Direct Test-Requirements
[1,4,5,1,4,5,1,2,3]	[4,5,1,2,3], [4,5,1,4], [1,4,5,1], [5,1,4,5]
[4,5,6,1,4,5,6,7]	[4,5,6,1,4], [1,4,5,6,7], [6,1,4,5,6], [5,6,1,4,5]
[4,5,1,4,5,6,1,4,5,6,1,4,5,6,1,2,3]	[4,5,6,1,2,3], [4,5,6,1,4], [1,4,5,6,1], [6,1,4,5,6], [5,6,1,4,5], [4,5,1,4], [5,1,4,5]

Testing-Paths	In-Direct Test-Requirements
[1,4,5,1,4,5,1,2,3]	[4,5,1,2,3], [1,4,5,1]
[4,5,6,1,4,5,6,7]	None
[4,5,1,4,5,6,1,4,5,6,1,4,5,6,1,2,3]	[4,5,6,1,2,3], [4,5,6,1,4], [1,4,5,6,1], [6,1,4,5,6], [5,6,1,4,5], [4,5,1,4], [5,1,4,5]

Table 4: 6 test paths are needed for Edge-Pair Coverage

Testing-Paths	Direct Test-Requirements
[1,4,5]	[1,4,5]
[1,4,5,6,7]	[1,4,5], [4,5,6], [5,6,7]
[1,4,5,1,4]	[1,4,5], [4,5,1], [5,1,4]
[1,4,5,6,1,4]	[1,4,5], [4,5,6], [5,6,1], [6,1,4]
[1,4,5,1,2,3]	[1,2,3], [1,4,5], [4,5,1], [5,1,2]
[1,4,5,6,1,2,3]	[1,2,3], [1,4,5], [4,5,6], [5,6,1], [6,1,2]

Table 5: 3 test paths required for Edge-Pair Coverage with prefix-graph algorithm

Testing-Paths	Direct Test-Requirements
[1,4,5,1,4,5,1,2,3]	[1,2,3], [1,4,5], [4,5,1], [5,1,2], [5,1,4]

[1,4,5,6,1,4,5,6,1,2,3]	[1,2,3], [1,4,5], [4,5,6], [5,6,1], [6,1,2], [6,1,4]
[1,4,5,6,7]	[1,4,5], [4,5,6], [5,6,7]

Testing-Paths	**InDirect Test-Requirements**
[1,4,5,1,4,5,1,2,3]	None
[1,4,5,6,1,4,5,6,1,2,3]	None
[1,4,5,6,7]	None

Table 6: 4 test paths required for Prime-Path Coverage

Testing-Paths	**Direct Test-Requirements**
[1,4,5,6,1,2,3]	[4,5,6,1,2,3], [1,4,5,6,1]
[1,4,5,1,2,3]	[4,5,1,2,3], [1,4,5,1]
[1,4,5,6,1,4,5,6,7]	[4,5,6,1,4], [1,4,5,6,7], [1,4,5,6,1], [6,1,4,5,6], [5,6,1,4,5]
[1,4,5,1,4,5]	[4,5,1,4], [1,4,5,1], [5,1,4,5]

Testing-Paths	**InDirect Test-Requirements**
[1,4,5,6,1,2,3]	None
[1,4,5,1,2,3]	None
[1,4,5,6,1,4,5,6,7]	[1,4,5,6,7]
[1,4,5,1,4,5]	None

Conclusion

The results and analysis of the comparison between different algorithms thus yields that prime path coverage when combined with prefix graph algorithm generates least number of yet effective test cases which can traverse all the states in less time. Other approaches such as node coverage or edge pair coverage also yields good results but the test path requirements are very large. Test Requirements which are traversed by test-paths using indirect routes are also been provided in the accompanying tables.

REFERENCES

[1] G. Sabaliauskaite, “Integrating Autonomous Vehicle Safety and Security,” no. level 0, pp. 75–81, 2017.

[2] F. Jiménez, *Future Perspectives and Research Areas*. Elsevier Inc., 2018.

[3] M. Hartmann, M. Viehweger, W. Desmet, M. Spitzer, M. Stolz, and D. Watzenig, “‘ Pedestrian in the Loop ’: An approach using augmented reality,” no. February, pp. 10–15, 2018.

[4] P. F. J. G. Wrage, “Reliability Validation and Improvement Framework Peter by CMU/SEI-2012-SR-013,” 2012.

[5] M. Mauritz, F. Howar, and A. Rausch, “Assuring the Safety of Advanced Driver Assistance Systems through a Combination of Simulation and Runtime Monitoring,” *Lect. Notes Comput. Sci.*, vol. 9953, 2016.

[6] J. Villagra *et al.*, *Automated Driving*. Elsevier Inc., 2018.

[7] K. Abdelgawad, M. Abdelkarim, B. Hassan, M. Grafe, and I. Gräßler, “A Scalable Framework for Advanced Driver Assistance Systems Simulation,” *Proc. 6th Int. Conf. Adv. Syst. Simul. (SIMUL 2014), Oct 2014, NIzza, Fr.*, no. c, pp. 43–51, 2014.

[8] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, “Testing advanced driver assistance systems for fault management with the VEHIL test facility ∗ Testing Advanced Driver Assistance Systems for Fault Management with the VEHIL Test Facility,” vol. 19, 2004.

[9] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, “Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations,” *Veh. Syst. Dyn.*, vol. 44, no. 7, pp. 569–590, Jul. 2006.

[10] F. Ambert, F. Bouquet, J. Lasalle, B. Legeard, and F. Peureux, “Applying an MBT Toolchain to Automotive Embedded Systems : Case Study Reports,” no. c, pp. 139–144, 2012.

[11] P. Kaur and A. K. Luhach, “An approach to improve test path generation: Inclination towards automated model-based software design and testing,” in *2016 5th International Conference on Reliability, Infocom Technologies and Optimization, ICRITO 2016: Trends and Future Directions*, 2016.

[12] P. Zech, P. Kalb, M. Felderer, C. Atkinson, and R. Breu, “Model-based regression testing by OCL,” *Int. J. Softw. Tools Technol. Transf.*, pp. 1–17, 2015.

[13] P. Kaur and R. Sobti, “Optimized MBT-Test Case Generation for Embedded System Controller Using LabVIEW and Sequence Graphs,” *Springer Adv. Intell. Syst. Comput. 709*, pp. 283–294, 2018.

[14] R. Kolagari *et al.*, “Model-Based Analysis and Engineering of Automotive Architectures with EAST-ADL To cite this version : Model-Based Analysis and Engineering of Automotive Architectures with EAST-ADL : Revisited,” 2018.

ISSN: 0971-1260

Vol-22-Issue-17-September-2019

[15] A. Aleti, “Designing automotive embedded systems with adaptive,” 2014.

[16] M. Tatar, “Enhancing ADAS Test and Validation with Automated Search for Critical Situations,” no. September 2015, 2016.