An Automated Testing Method for AUTOSAR Software Components Based on SiL Simulation

Sooyong Jeong, Woo Jin Lee
School of Computer Science and Engineering, Software Technology Research Center
Kyungpook National University
Daegu, South Korea
{kyo1363, woojin}@knu.ac.kr

Abstract—Testing automotive software using Software-in-the-Loop (SiL) simulation doesn't require the hardware such as test vehicles and ECUs. Existing testing techniques based on SiL rely on manual processes because automation techniques of testing AUTOSAR components using SiL simulation have not yet been established. In this paper, we present an automated testing technique and its implementation based on SiL simulation. We have evaluated the effectiveness and efficiency of the presented technique through a case study.

Keywords—software testing; AUTOSAR; Software-in-the-Loop simulation(SiL)

I. Introduction

Currently, vehicles have various electric/electronic devices that control and support stability, convenience, safety, and so on [1]. Software components in a passenger car contain millions of lines of code to include new technologies such as ADAS (Advanced Driver Assistance System) and ESC (Electronic Stability Control). Because of the increase in adoption of electronic devices in various areas of the automotive industry, testing of automotive software has become complex. To test automotive software in the AUTOSAR architecture, developers currently use Hardware-in-the-Loop (HiL) simulation or vehicle in considerable ratio [3]. However, the technique has its own limitations. For example, software components cannot be tested at early stages because the technique requires presence of hardware like ECUs and their specific configurations. On the other hand, Software-in-the-Loop (SiL) simulation techniques can be used to test automotive software components before the hardware configurations are specified [4]. The SiL simulation techniques take advantage of AUTOSAR [5] design concepts to simulate the behavior of software components by integrating the realistic environment simulation.

However, the existing testing techniques of automotive software components based on the SiL simulation do not support automated testing. Since both closed-loop and open-loop are included in the automotive system, it is hard to perform the automatic testing of automotive software. When the system is closed-loop and the environmental situation is changed, the sequence of input/output can be changed in unpredictable ways because of the feedback of the closed-loop systems. As a result, the testing process still depends on the manual operations.

In this paper, we present an automated testing technique for testing AUTOSAR software components based on SiL simulation. First of all, the notion of testing module which is automatically running the test cases and getting the results in the SiL simulation is proposed. Then, we elaborate on the working of the testing module in SiL simulation to test the software components automatically. To resolve the problem of testing closed-loop control system that is low predictability of expected result, the conversion of test cases for closed-loop control system in automotive is proposed. Finally, we perform a case study using our technique to show that it is able to test the automotive software components automatically and effectively.

II. Background

A. Closed-Loops of Automotive Control Systems

Control systems are divided into two categories, closed-loop and open-loop, depending on the existence of feedback from the output of the system. The closed-loop control systems use the previous output signals as input to make the desired output. In contrast, the open-loop control systems do not use feedbacks and their control action is independent of the process output [6]. In the automotive domain, majority of control systems are closed-loop that use various inputs including feedbacks. For example, ESC monitors steering, vehicle movement, and acceleration then adjusts brake and steering to stabilize the vehicle. In addition, the inputs from drivers also have closed-loops. For example, steering and acceleration should be varied by the targeting speed and vehicle configuration even if the driver is driving the car under the same road condition and course. Existing vehicle simulator tools support both closed-loop and open-loop driving control systems. However, because of the complexity arising from the combination of closed-loops in driving scenarios and control systems, the expected output may become unpredictable. The automated testing of such scenarios is difficult to implement.

B. Testing based on Simulation of Automotive Software Component

Since the automotive system has sophisticated structure, testing of automotive software has been done using a variety of testing methods. A large number of existing testing techniques

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) (IITP-2016-H8601-16-1002) supervised by the IITP(Institute for Information & communications Technology Promotion), and the BK21 Plus project (SW Human Resource Development Program for Supporting Smart Life) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (21A20131600005)

are based on HiL simulation and actual vehicle. In addition, the testing methods based on SiL simulation are used more and more recently for automotive software [3].

Although the testing techniques which uses an actual vehicle are the most accurate, they are dangerous and expensive. Therefore, HiL simulation is frequently used for testing automotive software instead of an actual vehicle. To test automotive software components using HiL simulation, the components are executed on HiL simulator hardware which provides virtual environments to test the software with actual ECUs, real-time processors, and I/O interfaces [7]. Since the technique uses actual ECUs, the software components should be configured for ECUs of the simulator device. The software components must also be compiled and in the executable form for ECUs to be tested in HiL simulation. It means that testing automotive software based on HiL simulation is hard at the early development stage because of ECU configuration and compilation. As an alternative to HiL simulation, various testing techniques based on SiL simulation have been proposed. For example, VEOS from dSpace [8] and ISOLAR-EVE from ETAS [9,13] tools are based on SiL simulation. Developers can test automotive software on these tools using virtual ECUs and virtual environment. However, in order to test automotive software in this way, developers should have additional knowledge of hardware to virtualize ECUs in the tool. Also, the software components should be configured for the virtual ECUs to be tested and run in the simulation tool. As a result, testing of automotive software cannot be performed in the early development stage which is a major limitation of existing testing tools.

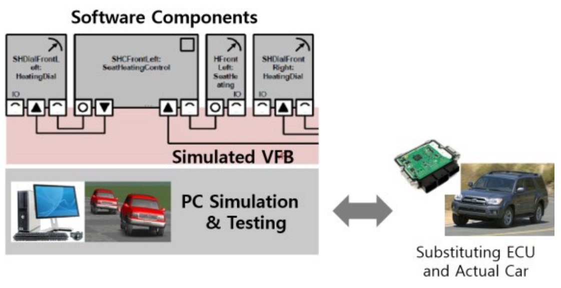

Software Components
SHDialFrontLeft: HeatingDial
IO
SHCFrontLeft: SeatHeatingControl
HFrontLeft: SeatHeating
IO
SHDialFrontRight: HeatingDial
IO
Simulated VFB
PC Simulation & Testing
Substituting ECU and Actual Car

Fig. 1. The Previous Work about Testing Based on Software-in-the-Loop Simulation for AUTOSAR Software Components

As a solution, we proposed a testing technique [4] based on SiL simulation using concepts of AUTOSAR. The technique can be used to test automotive software components in the early development stage. It uses the idea of simulating VFB layer [14] in the AUTOSAR architecture to run and test software components before the configuration for hardware is finalized. Unlike existing testing techniques, it minimizes the preliminary steps for testing since it doesn't require the ECUs configuration phase of AUTOSAR. Figure 1 shows the overview of our existing testing technique that could test automotive software components at source code level before the hardware configuration step.

However, our testing technique based on SiL simulation [4] is not automated. Not only our testing technique but also other existing testing techniques have the same problem. As noted by Altinger et al. [3], automotive testing still depends on manual testing globally. In consequence to this problem, in this paper, we have extended our existing work [4] by automating the test case generation and execution process for automotive software.

III. Test Automation Method

Our technique uses SiL simulation as a basis for testing automotive software. The SiL simulator in PC runs automotive software component code and calculates the result of the run and its effect on the vehicle virtually. From the result of simulation, we analyze and decide whether the software run properly and then we find faults and failures. However, the tasks of generating test cases and running the simulation was a manual process. Figure 2 shows the ordinary testing process based on SiL simulation. The test cases for simulation and source code of software components are put to SiL simulation. The log of the SiL simulation represents the result of vehicle simulation reflects the action and effect of the software components. The log is analyzed and compared with the expected output field of the test cases. Finally, a success/failure of software components are decided by analyzing the log.

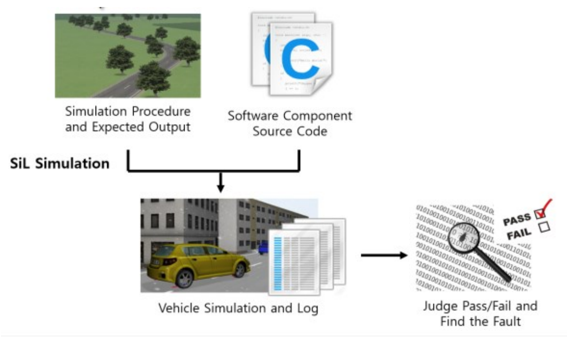

Simulation Procedure and Expected Output
Software Component Source Code
SiL Simulation
Vehicle Simulation and Log
PASS
FAIL
Judge Pass/Fail and Find the Fault

Fig. 2. The Testing Based on SiL Simulation for Automotive Software

In our existing testing method based on SiL simulation as described earlier, running and analysis of the test cases were done manually. To run the test cases and get their results automatically, we propose to integrate an automated testing module to the SiL simulation. Figure 3 shows the overview of our technique. The automated testing module is connected to the simulated VFB of simulation through an API. The main functionality of the module is to automatically run the simulation and make a decision on whether the output of simulation corresponds to the expectation. In the figure, gray circles represent the test cases for automated testing while white circles represent the output and result of the testing.

In this section, we present the implementation of automated testing feature of the SiL-based testing method for AUTOSAR software components. Figure 4 summarizes the approach of automated testing of this paper. (1) The test cases representing inputs of a vehicle driving scenario are converted to the temporal format which will be used in the automated testing. (2) The automated testing module is added to the testing method based on SiL simulation of our previous work [4]. The automated testing module gets the converted test cases as input and runs the simulation and tests of the code automatically.

A. *Test Case Conversion*

The test cases for automotive system has a characteristic that it includes closed-loops in both input scenario and control system. Due to closed-loops in the input scenario, changes in some input values affect the other inputs to preserve the consistency of the scenario. In addition, the closed-loops in the control system may cause changes in expected output due to changes in input. In other words, the input and output of automotive testing can be varied in unexpected ways by the feedback factors. Therefore, the analysis and prediction of the output for testing automotive control software is harder than other open-loop control software. Also, automating the testing process of automotive software is much harder. In this section, we present the conversion method of test cases to automate the testing of software components including closed-loops. The method gets the test cases in closed-loop format as input and converts it to the test cases in temporal format which is to be used in the automated testing method.

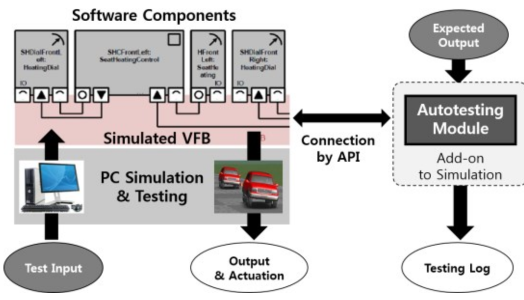


Fig. 3. The Attachment of Automated Testing onto the SiL Simulator for AUTOSAR Software Component

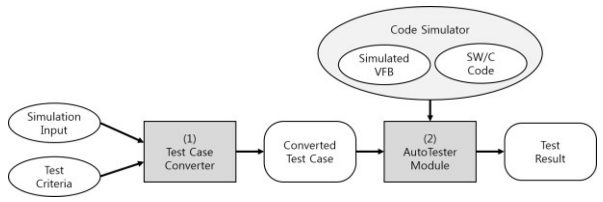


Fig. 4. The Implementation of Automated Testing onto the SiL Simulator for AUTOSAR Software Component

Figure 5 illustrates how the test cases including closed-loop are converted to temporal test cases. First of all, a developer extracts the logic and the algorithm from the specification and code of the control software to be tested. Because commercial vehicle simulators support using ‘additional data’ that controls the simulation during runtime, we used this feature of the simulator. The control software logic is converted to the ‘additional data’ for the vehicle simulation. The ‘additional data’ fields of vehicle simulator is capable to get the status of simulation, and give a control to the simulation according to the status. In short words, it can work like a software component built in the car. The test input is derived from the design such as code and specification consist of simulation driving procedure, to run and make stimuli of the software components. Then they are combined into a simulation input of vehicle simulator. In the combination, the converted software logic in ‘additional data’ will affect the input and output of a vehicle simulation during runtime and be a guideline of success/failure of the testing. Running a preliminary simulation with the combined input data, the developer gets the temporal data that contains inputs versus time and outputs versus time from the log. The data are used by the developer as the expected output.

Then, the expected output extracted from the preliminary simulation is reformed by using classification tree and partitioning [11, 12]. Once the inputs of the system are partitioned by making the classification tree, a combination table is formed from the temporal test data. The partition of the classification tree divides the test data into rows of combination table. If an input makes a transition of the classification tree, a new row of combination table representing a single test case is generated. The processes are done recursively to complete the combination table and it acts as a unit of separating the continuous testing sequence to discrete one. Finally, the set of test cases from the combination table and temporal test input data are used for automated testing.

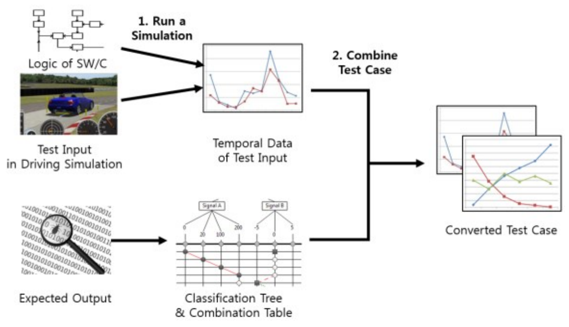


Fig. 5. The Detail of Feature of Test Case Conversion

The conversion process contributes in making test cases flexible for closed-loops since it can automate generation of test cases to expectable form. When the simulation input of the closed-loop changes, we can revise the test cases easily and get new expected output by the proposed conversion.

B. *Automated Testing using SiL Simulation*

The existing testing method based on SiL simulation has focused on enabling the testing early without the actual system. However, the process of automated testing has not yet been established and it heavily depends on manual operations performed by developers. Therefore, we automate the testing method based on SiL simulation by adding an automated testing module.

To make the testing process automated, the temporal test cases generated in the proposed conversion method are used. The test cases are separated into two files: test input and expected output and they are used throughout the testing process. The test input is put to the vehicle simulator directly to run a simulation according to the scenario. The expected output of the test cases are put to the automated testing module. In the automated testing module, output is monitored at the specified timing at every rows of the combination table and it is compared with the expected output from the test cases. During comparison, if the actual output is mismatched with the expected one, the failure is written to the test report. Otherwise, the success is written to the report. Figure 3 shows the addition of the automated testing module to the simulated VFB. Figure 6 and 7 show the pseudocode of the automated testing module and its connection with the simulated VFB. The simulated VFB provides an interface for communication among AUTOSAR software components, the API of Windows PC, and the vehicle simulator to run the automotive software components. In Figure 6, there is a function named 'Simul_proc()' that processes the action of sensor/actuator of the vehicle simulator. It is called in every iteration of simulation and reads the current status of simulation then redirects to the branch for the current simulation status. The code calling the API of the automated testing module is added inside the 'Simul_proc()' function of the simulated VFB.

```
                    (In Simulated VFB)
function Simul_proc (time)
  if (initialization of simulation)
    …
    call Init_testingmodule () /* to be added */
  end if
  if (the start of iteration)
    …
  end if
  if (the end of iteration)
    if (auto testing is activated)
      call Test_timestamp (output, time) /* to be added */
    end if
  end if
  if (termination of simulation)
    …
    call Print_result() /* to be added */
  end if
end function
```

Fig. 6. The Connection to the Auto Testing Module from Simulated VFB

Figure 7 shows the pseudocode of the automated testing module in detail. First, the variables to be used globally are defined. The function 'Init_testingmodule()' loads and parses the file of expected output and the combination table then the variables are initialized and the module is ready to run automated testing. In the main loop of simulation, the simulator gets the input and calculates at the start of iteration and makes outputs at the end of loop. Therefore, we add the test function 'Test_timestamp()' to the end of the iteration. It hooks the set of outputs and the current timestamp then passes them to the automated testing module. It compares the actual output with the expected output to test whether the software component controls the vehicle properly in the simulation. The timing of comparison is decided by the timing field in the combination table. The result of the timestamp including success/failure and its detail value are written to the test report file. Running the iteration throughout the simulation, these activities are iterated and the test results are accumulated to the test report file. Finally, in the case of termination of simulation, the accumulated test report is printed and the testing is ended.

IV. Case Study

The proposed testing method is used to test an auto-braking system that recognizes the object in front of car and performs emergency braking. The auto-braking system gets the input from sensors and lidars then controls the brake master cylinder to stop the vehicle. The power of the master cylinder should be adjusted throughout the driving according to the bearing angle and distance from recognized object. Initially the system and software was designed for golf cart by the other laboratory of Kyungpook National University. In the testing, the system is applied to an A-segment small passenger car simulated by the vehicle simulator. Then, we tested whether the software component is working properly using the SiL simulation.

Figure 8 shows the principle of auto-braking software used in the golf cart and its demonstration of testing based on SiL simulation proposed in this paper. Using the lidar, the system and software detects the object. The zone marked (a) named danger zone is with the proximity less than 7m and bearing angle -40° ~ -20° and 20°~ 40°. The zone marked (c) named emergency zone is with the proximity less than 5m and bearing angle -20° ~ 20°. The zone marked (b) is with the proximity 5~7m and bearing angle -20° ~ 20°. When the vehicle encounters the object and the lidar sensor of the cart detects it, the auto-braking software controls the cart. If the sensor detects the object at (a), it decelerate to 30% of the current speed. If the sensor detects the object at (c), it activates brake fully to stop the vehicle immediately. If the sensor detect the object elsewhere, the auto-braking software doesn't do anything.

```
                 (In Automated Testing Module)
timing[m] ← timing of the mth row of the combination table
expect_value[m] ←     expected value of the nth field in the mth
                      row of the combination table
threshold ← the threshold on error of field n (optional)
test_result[m] ← the test pass/fail in the mth row of the
                 combination table
index ← the current index of the row of combination table
function Init_testingmodule ()
    load expected output file
    parse expected output file
    fields, expected_value, threshold are initialized
    index = 0;
end function
function Test_timestamp (output, current_time)
    output ← the output from the simulation
    current_time ← current timestamp of the simulation
    if (current_time is equal to timing[index])
        if (output is equal to expect_value [index] within
        threshold)
            write the test_result[index] successful and its detail
        end if
    index = index + 1;
    end if
end function
function Print_result()
    loop for each test_result
        write test_result[i] to file
    end loop
end function
```

Fig. 7. The Pseudocode of Automated Testing Module

To test automotive software components based on the method presented in [4], we used CarSim from Mechanical Simulation [10] as the vehicle simulator. We make an input of a test case for CarSim that presents the driving of a vehicle. It consists of vehicle configuration, driving control (speed control, steering control, gear shift control), road condition. First of all, the vehicle configuration includes the powertrain, brake system, steering, sensors is modelled by us. We used the default configuration of A-segment small passenger car provided by CarSim and additional lidar sensor customized. The logic of the software component was written in CarSim event data and combined with the vehicle configuration. In the consequence, they models the car with the auto-braking system working without fault. After the vehicle configuration, we made a CarSim scenario that simulates the situation of car facing a tree while driving, it has a role of input of the test case. Since the input and the software component have closed-loops, we ran the preliminary simulation with the data and did a conversion of test case. Finally, we got the temporal test cases and its combination table from the conversion. Figure 9 indicates the test cases.

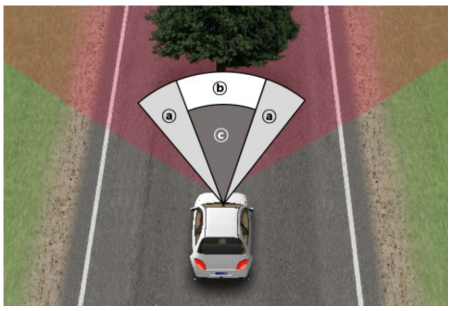
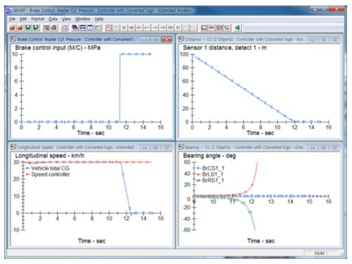
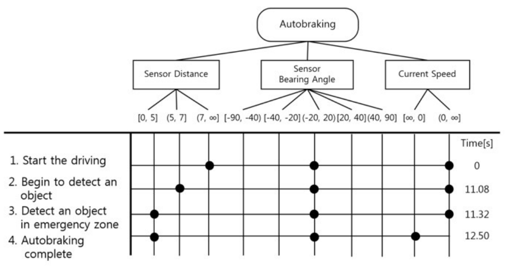


Fig. 8. The Principle of Autobraking System under Test and Demonstration of Testing based on SiL Simulation.


Brake control input (M/C) - MPa
Sensor 1 distance, detect 1 - m
Longitudinal speed - km/h
Bearing angle - deg
Time - sec
Time - sec
Time - sec
Time - sec

(a) Converted Test Input in Temporal Data



(b) Classification Tree and Combination Table for Test Data

Fig. 9. A Test Case for the Auto-braking Software

After the establishment of the test cases, we ran a SiL simulation of them with the auto-braking software code and tested it with the test cases shown in Figure 9.

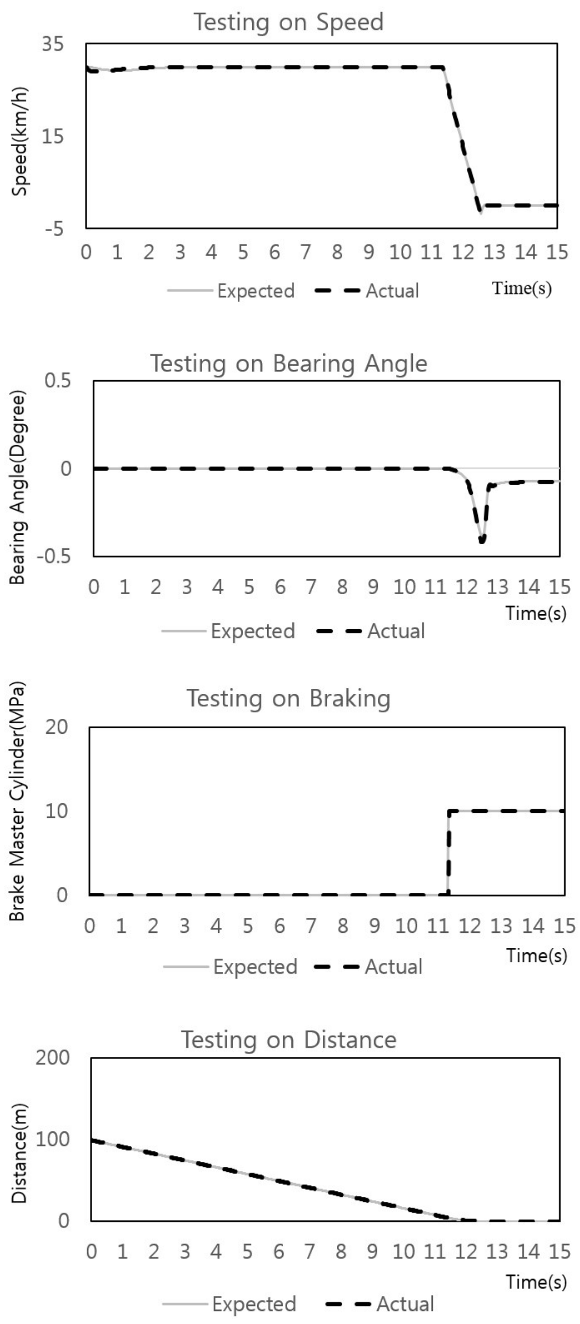


Fig. 10. A Test Cases and Result for Auto-braking Software based on SiL Simulation

Figure 10 shows the results of testing. In the figure, the gray solid line means the expected output got from the test case conversion, and black dotted line means the actual output from the simulation. To show that we used the combination table to judge the test discretely from the continuous data, we marked the numbers of test cases on the graph. (1) The testing based on

SiL simulation is started. (2) When the object appears closer than 7m, the lidar sensor begins to detect the object. According to the bearing angle, the auto-braking is controlled. In this case, when the sensor bearing angle is between -20° and 20° and the distance is between 5 and 7m, no braking is activated although the sensor detects the object. (3) When the distance to the object gets closer than 5m preserving the sensor bearing angle between -20° and 20°, the auto-braking software regards that it is in 'emergency zone'. Then the software performs braking to stop the vehicle. (4) At the end of the testing, the vehicle stopped in front of the object and successfully avoided the collision. The brake master cylinder holds to avoid the movement of the vehicle. In the test cases, we were able to see the success of all the test cases from the data. The numerical errors of simulation between expected result and actual result are also acceptable. To measure the error rate, we used Equation (1). In the equation, *n* means the number of samples of the continuous input. E_i is the expected value of *i*th sample. A_i is the actual value of *i*th sample. D_i is a difference in percentage between the expected value and the actual value at *i*th sample. It assumes that E_i cannot be zero and be ignored even if it is zero. The difference is then written in average of every samples of D_i.

$$D_i = \begin{cases} \left|\frac{E_i - A_i}{E_i}\right| \times 100 & (E_i \neq 0) \\ 0 & (E_i = 0) \end{cases}$$

$$Difference = \frac{\sum_{i=0}^{n} D_i}{n} \tag{1}$$

Using the equation in the testing, the error rate of brake master cylinder pressure was about 0.166%, the error rate of distance was about 2.588%, and the error rate of speed was about 0.744%. So, we conclude in this test that the auto-braking software components are made sufficiently precise.

This case study shows that the proposed testing technique based on SiL simulation can be successfully used to test automotive software in early stage. The results also prove that the technique is effective and efficient.

V. Conclusion

We have discussed the current state of the testing techniques for automotive software and their limitations. Since many automotive software involve closed-loop controls and feedbacks, automated testing of automotive software has been a difficult task. Therefore, the manual testing has prevailed in the automotive testing globally. To resolve the problem, we proposed the automation of the software testing method based on SiL simulation [4]. The concept of integrating an automated testing module to the existing SiL simulator has been presented which provides test results from the SiL simulation automatically. In order to use the automated testing module, the test cases including the closed-loops are converted to a temporal form. A test case conversion method using vehicle simulation has been presented and the converted test case is combined with classification tree method. We also performed a case study using our automated testing technique to test existing auto-braking software in the vehicle. The results show how the test cases are converted and how the testing is done in practice. Through the case study, we also showed the efficiency and effectiveness of our proposed testing method.

References

[1] K.Grimm, "Software Technology in an Automotive Company – Major Challenges," in Proceedings of the 25th International conference on Software Engineering, pp.498-503, 2003.

[2] R. N. Charette, "This Car Runs on Code," in IEEE Spectrum, vol. 46, no.3, 2009

[3] H. Altinger, F. Wotowa, M. Schrius, "Testing Methods Used in the Automotive Industry : Results from a Survey," in Proceedings of the 2014 Workshop on Joining AcadeMia and Industry Contributions to Test Automation and Model-Based Testing, pp. 1-6, 2014

[4] S. Jeong, Y. Kwak, W. Lee, "Software-in-the-Loop Simulation for Early-Stage Testing of AUTOSAR Software Component," in Proceedings of the 9th ICUFN, pp. 59-63, 2016

[5] AUTOSAR, [Online]. http://www.autosar.org

[6] K. Ogata, "Modern Control Engineering"

[7] Hardware-in-the-Loop (HIL) Test System Architectures. [Online] Available:http://www.ni.com/white-paper/10343/en/

[8] dSpace VEOS. [Online] Available:https://www.dspace.com/en/inc/home/products/sw/simulation_software/offline_simulator.cfm

[9] ETAS ISOLAR-EVE. [Online] Available:http://www.etas.com/en/products/isolar_eve.php

[10] Mechanical Simulation CarSim. [Online] Available:https://www.carsim.com

[11] M. Grochtmann, K. Grimm, "Classification Trees for Partition Testing," Software Testing, Verification and Reliability, Vol. 3, 1993

[12] M. Conrad, "A systematic approach to testing automotive control software," in Proceedings of Convergence, 2004

[13] Muli, M. and Cassar, J., "Virtual Validation - A New Paradigm inControls Engineering," SAE Technical Paper, No.2013-01-2404, 2013.

[14] N. Naumann, "Autosar runtime environment and virtual function bus," Hasso-Plattner-Institut, Tech. Rep, 2009.