Automating the Evaluation Process of Software Testing in Vehicles

Linh Nguyen Mai

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6098
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

The Safe Driving Test (SDT) is a software test performed in a real vehicle where the software is flashed to an ECU and connected to the CAN network in the vehicle. The test driver then conducts different driving scenarios according to the test instructions of the SDT. All the data during testing is logged and then evaluated manually by a test engineer. The entire process including testing, evaluating and fixing defects is very time-consuming and ineffective, especially the evaluation. Therefore the possibility of automating the evaluation process in Safe Driving Tests was explored in this master thesis.

The result of this work was a program containing an automatic evaluation process for the test cases in the Safe Driving Test. The program takes log files of the test cases as input and generates a report containing evaluations of each test case as output. This program can be integrated into different projects at BorgWarner.

The conclusion of this work is that it is definitely possible to automate the evaluation process and an automatic evaluation process will save time. However the test results must still be verified manually by a test engineer since the evaluation can only provide a pass or fail result and not the cause of the failed results.

Acknowledgements

This master thesis was done with support from BorgWarner in Landskrona. I would like to express my gratitude to Måns Andersson, supervisor of the Software Testing Department at BorgWarner, who came up with this thesis proposal and supported me with technical information during the work. Thanks to Mattias Wonizak, my supervisor at BorgWarner for his guidance and valuable input throughout the project. Thanks to Karl-Erik Årzén, my supervisor at LTH, for his feedback and proofreading. Thanks to Anton Cervin for being my examiner and for his feedback.

Contents

1. Introduction **9**
1.1 BorgWarner 9
1.2 Problem Formulation 10
1.3 Objectives and Approach 11
1.4 Delimitation 12
1.5 Related Work 12
1.6 Terminology 14
1.7 Outline 15

2. Background **16**
2.1 Software Testing at BorgWarner 16
2.1.1 Software Testing in Simulated Environments 17
2.1.2 Software Testing in Real-world Environments 19
2.2 Safe Driving Test 19
2.3 Front cross differential technology 20

3. Equipment **21**
3.1 CANoe 21
3.2 vTESTstudio 22
3.3 CAPL 22
3.4 Log file 23
3.5 Vector Hardware 23
3.6 Environment Setup 23

4. Software Development **24**
4.1 Approach 24
4.2 Requirements 25
4.3 Software Design 27
4.4 Software Implementation 28
4.4.1 Log File Preprocessing 28
4.4.2 Generic Solution for Customer-Specific Configurations . . 29
4.4.3 Algorithms 30
Accelerator Pedal Lift Detection 36

Moving Average 36
Delay Determination Between Signals 37
Similarity Comparison Between Signals 40

5. Verification and Result 42

5.1 Verification 42
5.1.1 Evaluation Interval Detection 42
5.1.2 Evaluation Process 46
5.2 Result 49

6. Conclusion and Future Work 53

Bibliography 55

1
Introduction

Most vehicle systems nowadays are operated under control of software; therefore, it is critical that the software is tested thoroughly to ensure that the software meets the safety requirements and operates safely in real life driving. Software verification is an indispensable part in vehicle software development. Since the software verification is a time-consuming process BorgWarner has therefore put a lot of effort into automating the tests as much as possible. Automation of test cases makes it possible to run thousands of test cases in very short time and thus shortens the verification time. The purpose of this master thesis is to automate the evaluation process of a test conducted at BorgWarner.

This chapter provides an introduction to BorgWarner as a company and defines the problem that will be solved along with the approach used to solve it. The works related to this thesis will also be presented and discussed in this chapter.

1.1 BorgWarner

BorgWarner is a multinational corporation and a large supplier of components and parts for the automotive industry. The company has a total of 68 facilities across 19 countries [BorgWarner, 2019]. Subsidiary companies of BorgWarner are divided into two groups: Engine Group and Drivetrain Group. BorgWarner specializes in powertrain products for different technologies such as combustion, hybrid and electric powertrains [BorgWarner, 2019].

The practical work of this thesis was conducted at BorgWarner PowerDrive Systems (PDS) in Landskrona, the European headquarter of BorgWarner and one of their 21 locations with PDS operations.

1.2 Problem Formulation

A very important part of the overall vehicle system today consists of software and other electronic components. As with any other part of the vehicle it is necessary to test and verify that it can operate safely in real-life situations.

At BorgWarner the software is tested automatically and/or manually. Automatic testing is testing that can run without any human interaction during the testing. In automatic testing both test stimuli and evaluation are autonomous. Since everything is automated the software is always tested in a simulated environment. Manual testing is testing that requires the interaction of a test engineer during testing. The interaction can be in the form of providing the required test stimuli or evaluating the test result or both. Most of the testing at BorgWarner nowadays is automatic, but there is still a small portion of manual testing.

In addition to performing automatic testing of the software in a simulated environment, the Software Testing Department at BorgWarner also tests software in real and dynamic driving environments to cover test cases that cannot be covered by automatic testing. Tests conducted in real driving environments are called Safe Driving Tests (SDTs) and the usual test procedure of the SDTs is as follows:

- The software to be tested is uploaded to an ECU (Electric Control Unit), which is then installed in a car. The car is then taken to a test track where every test case is performed manually. The test results are not analysed thoroughly during the test, but all measured signals from the car are logged.
- Proper analysis of relevant signals such as the speed of the motor, the acceleration, the control angles, the signals from the electronic stability program etc. is done later by a software evaluation engineer and, the results are evaluated thoroughly.

Normally, it takes three to five days to perform the analysis and summarize the test report. If any defects are found, the software is sent back to the developers to be fixed, and then the whole procedure will be repeated again until the software is approved (see Figure 1.1). The entire process, including performing the test cases, evaluating and fixing defects, takes approximately two weeks. Besides the cost in terms of time taken for evaluation, the company must also pay other expenses such as renting a test track, wear and tear in the car, risk that something happens during testing and so on. However, the most important factor is that in the manual evaluation the result depends a lot on the experience, opinion and accuracy of evaluator.

BorgWarner wants to improve the workflow in the described process above with the help of automatic analysis and a simulated model of the car. By testing the software thoroughly with the help of the simulated model before testing in a real vehicle some problems can be excluded, such as the software is not working at all when it is time to test in the real driving situation. Automating the evaluation process would also reduce the time and cost of the testing process and simplify the process of finding out the cause of unexpected results. With automatic evaluation all the test results will be assessed in the same way regardless of the evaluator.

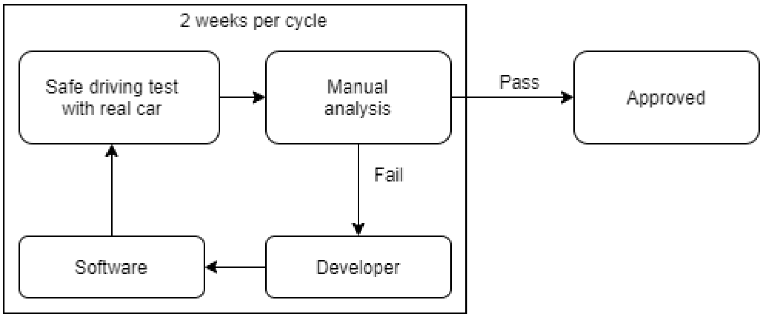


Figure 1.1: Overview of the present workflow.

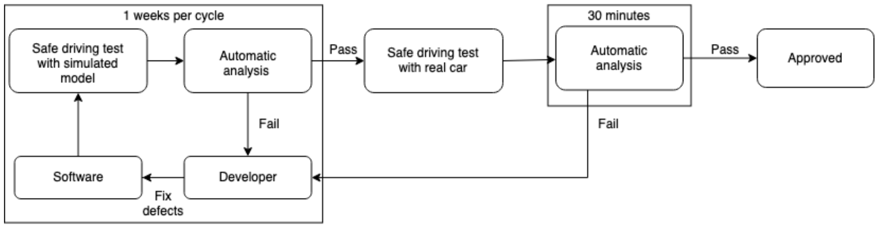


Figure 1.2: Overview of the new planed workflow.

1.3 Objectives and Approach

The main purpose of this master thesis is to implement a software evaluation program which analyses and evaluates vehicle software automatically based on data logged during testing. The logged data can be produced either from a simulation or from a test performed on a test track.

Using the software evaluation program together with the vehicle simulation will transform the current workflow in the SDT (see Figure 1.2). Before performing the tests on a test track the software will first be tested with the help of a simulated vehicle model until all found defects are fixed. After that the software will be tested in a test track for a final evaluation.

If successful, this would improve the efficiency of the analysis significantly. The software evaluation program therefore has two purposes: analyzing the test results from the simulation and analyzing the test results from the test track.

The successful project should be able to answer the following questions:

- Does BorgWarner save time by automating the analysis process?
- Can BorgWarner rely on the test result provided by the evaluation program?
- Does BorgWarner get the same test result from automatic evaluation vs manual evaluation?

The overall approach consists of the following parts:

- Get a good understanding of how the manual analytic work is performed.
- Investigate how the manual analytic work can be automated.
- Investigate how the software should be designed to maximize the reusability of the evaluation program in different projects.
- Design and implement the evaluation process for each test case.
- Evaluate the accuracy and reliability of the evaluation program.

1.4 Delimitation

Although the future vision of BorgWarner is to automate the SDT using the simulated vehicle model together with the evaluation program, this master thesis is limited to the implementation of the evaluation program. It is not within the scope of this master thesis to develop the simulated vehicle model or evaluate test results provided by the simulated vehicle model. There is already a complete implementation of a simulated vehicle model developed by BorgWarner[Espfors, 2018].

1.5 Related Work

Since this master thesis is about automating the evaluation process, the related works that have been considered are those that fall into the category of automatic testing and evaluation. Automatic testing and evaluation of software is neither a new concept in software development in general nor in software development in the automotive industry. It has been widely applied in software development and therefore there are already a lot of research and many applications done in this area. The related works that have been done at BorgWarner and two related works conducted outside BorgWarner, which are considered as most relevant to the master thesis, will be reviewed in this chapter.

The software testing department at BorgWarner has made a huge effort in test automation. Approximately 95 percent of the test cases are automated and conducted in a simulated environment. Despite the high percentage of automated test cases, attempts to automate the evaluation of test results from real world testing have not been explored yet. The reason is probably the difference between test stimuli provided in simulated environment and test stimuli provided in real world. In a simulated environment the test stimuli are mainly static, meaning that the test conditions are the same throughout the test. An example of static test stimuli can be setting the wheel speed to 20 km/h. The wheel speed in a simulated environment will always have the same value throughout the test. On the other hand, in a real world testing the test stimuli is dynamic, which means the wheel speed will not be exactly 20 km/h throughout the test. The wheel speed varies because there are many things that affect the wheel speed, like how well the driver can hold the accelerator pedal in the same position or the uneven drive surface can reduce the wheel speed, etc. Dynamic test stimuli lead to dynamic test results. Due to the dynamic test results, it is difficult to create a reliable evaluation program.

Although, there are many studies and applications relating to automatic testing and evaluation, there is very little research which is specifically done in automatic evaluation of dynamic testing. In the following paragraphs two thesis works that are similar to the assignment will be reviewed.

[Conrad et al., 2005] has developed a tool called MEval, an automatic evaluation program designed for ECU software tests. This program can be used to evaluate the test results which are time-dependent signals. The idea behind MEval is to evaluate the test result based on signal comparisons. MEval takes the approved test result (signal data) from previous tests and the current result signal as inputs. The program then evaluates the current result signal by comparing it with the approved test result, which is used as reference data. The signal comparisons are conducted in two steps: preprocessing and comparison. At the first step the signals will be preprocessed using a difference-matrix which is a matrix where entries are absolute values of the signals differences $DiffM(t_i,t_j) = |o(t_i) - o`(t_j)|$, $o(t_i)$ is the value of reference signal at sample $i$ and $o`(t_j)$ is value of the current output signal at sample j. The purpose of this step is to adjust the current result signal to the reference signal so that the signals will be time aligned as well as possible. If the test result passes the first step it means the time deviations do not violate a given threshold. MEval will then proceed to the second step which is a comparison of the signals using relative and slope-dependent differences methods. The output of MEval is a single value which shows how much the signal in the test result differ from reference data. If the value is below the maximum accepted deviation the test result passes the evaluation.

The idea behind the master thesis performed by [Adenmark, 2003] at Scania was to automate the integration tests at Scania as much as possible. This included automation of the test stimuli as well as automation of evaluation of test result. The result of this master thesis was a program containing several automatic tests. This program provided the required stimuli for each test and then evaluated the responses from the ECU. While the program goes through and performs each test case one by one it also fills in the test protocols with test results. Only the evaluation part of this program is of interest since it is the focused area of my master thesis. Both the evaluation in automated CAN communication test and User Function test implemented by [Adenmark, 2003] work similar. The program evaluates the test result by extracting information from response messages that are of interest and compare the extracted information with the requirements provided in the test protocols. The test result and any deviations that might occurs will be automatically filled into the corresponding test result field of the specific test in the test protocols.

Although the works reviewed above are similar to this master thesis, these solutions cannot be applied directly to solve this assignment. The thing in common between the solution provided by [Conrad et al., 2005] and this work is that both programs evaluate test results by analyzing the resulting signals collected from the test. However, what makes this work different from his work is that the program is expected to evaluate signals against the requirements instead of former approved data. The evaluation must be independent from former test result, because there might be a newly developed function that has never been tested and therefore do not have any former data as reference. In [Adenmark, 2003] the evaluation is limited to binary evaluation which means either the test result is identical to the expected result or not, while the evaluation program in this master thesis should also be able to evaluate non-binary cases like evaluating the value trend of the signals if it is continuously increased or decreased. Even though these works somewhat differ from this assignment they have been used as inspiration for this thesis work.

1.6 Terminology

The following terminology will be used throughout the thesis:

Evaluation program in this context means a program consisting of evaluation processes for multiple test cases.

Test case in this context contains a driving scenario and expected results. By performing the driving scenario and comparing the outcomes with expected results the tester will be able to determine whether the software under test fulfills the requirements or functions correctly.

1.7 Outline

The rest of this thesis is outlined as follow: Chapter 2 provides the reader with information about the software testing at BorgWarner and a description of the manual evaluation of Safe Driving Tests. The environment setup and a brief description of each component are given in Chapter 3. The approach for solving the problem followed by explanations of the algorithms used in this assignment is presented in Chapter 4. Chapter 5.2 presents the verification process of the software and the result of this master thesis. Lastly, the conclusion and remaining problems that should be solved or improved in the future are given in Chapter 6.

2

Background

This chapter aims to provide readers with the necessary information to understand how the software evaluation process is performed at BorgWarner, Landskrona and what software is used as a starting point of this project. The software evaluation process is done in two parts, simulation-based testing and real-world testing. Section 2.1 describes the simulation-based testing and real-world testing. The Safe Driving Test is described in Section 2.2 and Section 2.3 provides a description of the FXD technology.

2.1 Software Testing at BorgWarner

Products developed by BorgWarner have to undergo a strict testing process to ensure that they operate as expected, while also meeting the functional and safety requirements demanded by customers and international standards before they are released. Serious defects in a product can cause unwanted accidents that cost human life, therefore BorgWarner and any supplier within the automotive industry has put a lot of focus and effort on testing.

The software development process in a project at BorgWarner is an iterative process and can be described as a variant of the V-model [Petersson, 2014] which is illustrated in Figure 2.1 below. These phases are gone through for each iteration. In a project there will be a team from the software development department works together with a team from the software testing department. The development team is responsible for phases on the left and Software Unit Test while the software test team is responsible for the remaining phases on the right. The two teams work together and go through these phases in parallel. Requirements of the software are divided in work packages which are to be implemented and tested in each iteration. In each iteration the development team works on the implementation and unit testing, while the test team designs and implements test cases. In that way, the software test team is ready to run tests once a delivery is made.

A variety of tests are carried out by the software testing department. In this section a description of the software testing part that concerns the master thesis is provided. The software testing in this context can be divided into two types: software testing in simulated environments and software testing in real world environments. Both of these types are black-box testing, which means that only the functionalities of the software are examined, not its internal structure or code.

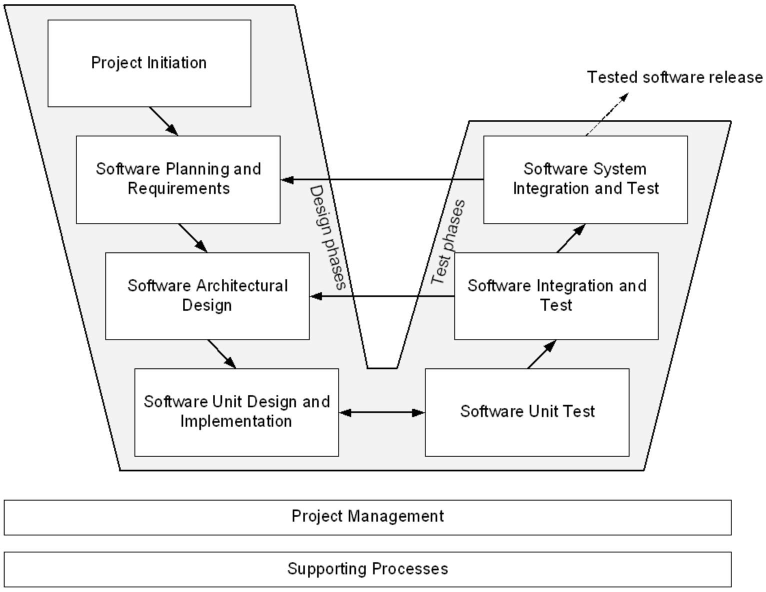

Project Initiation
Tested software release
Software Planning and Requirements
Software System Integration and Test
Design phases
Test phases
Software Architectural Design
Software Integration and Test
Software Unit Design and Implementation
Software Unit Test
Project Management
Supporting Processes

Figure 2.1: Phases in software development process at BorgWarner PDS Europe [Petersson, 2014]

2.1.1 Software Testing in Simulated Environments

The software testing department at BorgWarner in Landskrona is responsible for testing software developed for individual vehicle manufactures. The software to be tested here is the software that operates in an ECU. Each vehicle manufacturer has different communication conventions and requirements on the behavior of the software. Therefore, each project has the potential to be unique. Software at Borg-Warner can be tested manually and/or automatically in a simulated environment. For most of the cases the software is tested automatically, but sometimes manual tests are carried out.

The simulated environment here means that the ECU is connected to a system which operates like a real vehicle and the ECU cannot differentiate between a real vehicle environment and a testing environment. This is achieved by using CANoe (see Section 3.1) and hardware produced by Vector. CANoe simulates the rest of the network nodes while Vector hardware simulates the sensors and actuators. Figure 2.2 shows an example on how the ECU, CANoe and Vector hardware are connected to each other to create a testing environment.

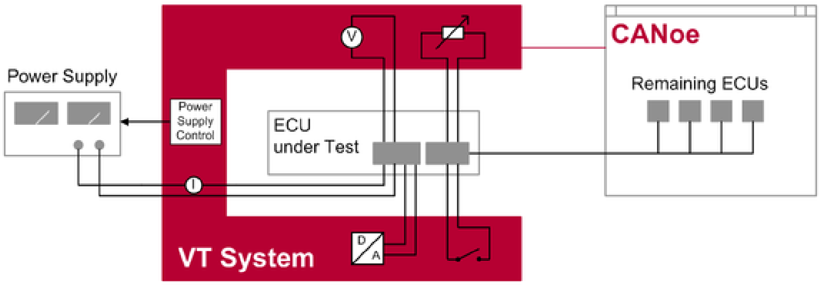


Figure 2.2: Testing of the ECU in simulated environment [Vector, 2019c].

All the test rigs used at BorgWarner are built on Vector products and can be categorized in two types: VT System and VN System. The difference between them is that the VT System is more powerful and offers additional functionalities that can be used to test hardware related errors such as short-circuits, pump error etc, while VN Systems are more suitable to test software related errors and diagnostic communications.

Test stimuli and error injections can be done automatically via CANoe to the test rigs. The test rigs are capable of for instance creating short-circuit, or open circuit by switching relays or setting desired temperatures. The capability of the test rigs makes it possible to run thousands of test cases automatically without the assistance of a human being.

With the help of VT and VN Systems tests can be performed completely automatically, from providing required test stimuli to evaluation. In automatic tests the test sequences are run automatically, and at the end of each test a test result is produced. The test engineers' job after a test run is to go through the test result and investigate the cause of the failed test cases. The test engineers also perform manual testing in case there are test cases that has not yet been implemented or are impossible to automate. Manual testing requires interaction between the test engineer and the simulated environment to generate inputs for the software and to verify or investigate the outputs provided by the software.

2.1.2 Software Testing in Real-world Environments

In order to be certain that a product operates well in a simulated environment also performs well in real-world environments, testing of the software in real world environments ought to be conducted as a final check before deploying to production. There are some conditions in the real world that sometimes cannot be simulated, such as the state of the road (wet, dry, icing), terrains with different surface (asphalt, grass, rocks, icy), etc so real world testing should be performed to cover driving situations that cannot be covered by testing in a simulated environment. This is also a chance to verify that the software operates and collaborates properly with the rest of the vehicle.

For this purpose, test tracks in different climates are used. There are two test tracks located in Sweden, one in Arjeplog in the north and the other in Ljungbyhed in the south. The test track in Arjeplog offers a cold climate and icy roads for testing the behavior of the software in different driving scenarios when it is exposed to tough climate and terrain. The test track in Ljungbyhed offers, however, a warmer climate and asphalt roads.

2.2 Safe Driving Test

As mentioned earlier, in addition to software testing in simulated environments the Software Testing Department at BorgWarner also conducts testing of the software in real world environments. The SDT is one of these tests which belongs to testing in real world environments. The test is often performed at the end of the development cycle to verify that besides normal operations the software also manages to handle extreme driving situations safely.

Since the test has been conducted for many years there are already well-created requirements and test specifications for test engineers to follow. The requirements specify the expectations BorgWarner has on the functionality of the software regarding safety, while the test specifications describe how the test cases should be executed and what the expectations on each test case are.

To conduct the SDT, the test engineer mounts the ECU flashed with test software to a vehicle. A PC is connected to the Vector hardware, which in turn is also connected to the CAN network that the ECU is connected to. By including Vector hardware and CANoe (see Section 3.1) to the network the test engineer manages to monitor and log the signals and behavior of the software.

At the test track the test engineer performs each test case one by one by following the test instructions in the test specification and quickly verifies that the corresponding expectations on the test case are fulfilled or by taking notes if there is any deviation. This procedure is very time-consuming and exhausting for the driver since the driver can get motion sickness by conducting many extreme driving scenarios.

After performing the test cases, the next step is evaluation. The test engineer does a thorough examination of the test results by analysing the logged data. If any deviation or defect is found the test engineer collaborates with a developer to find out the cause. As mentioned above this process is the most time-consuming part of the testing which takes 3 to 5 days plus approximately 2 weeks for the developers to fix the defects if any. This process continues until the software is approved. Figure 1.1 shows a summary of this process.

2.3 Front cross differential technology

As a starting point for the requirement elicitation and implementation of the software I studied the requirements, test instructions and log files of the software developed for BorgWarner's electronic limited slip differential (eLSD) technology called Front Cross Differential (FXD) [BorgWarner, 2015]. This technology is designed to improve traction, handling and stability of front-wheel drive (FWD) vehicles. The eLSD with FXD technology is installed behind the gearbox in a FWD vehicle. FWD vehicles often face an common problem which is the difference in levels of traction between the driving wheels at cornering and acceleration. The difference in levels of traction causes the vehicle to have a tendency to turn toward the direction of the wheel with lowest traction.

This problem can be overcome by using the FXD technology. The FXD technology uses data of the steering wheel angle, engine torque and yaw rate to calculate and adapt the amount of torque needed for different driving situations. To prevent the wheel slipping from occurring the FXD controls the locking torque between the front wheels and applies the power to the wheel with best traction. To improve driving performance during cornering the FXD technology reduces the inner wheel slip by providing more power to the outer wheel. The stability of the vehicle is improved by sending torque-vectoring effect at understeering and yaw-damping effect at oversteering.

3

Equipment

This chapter provides a brief introduction to the software and hardware used to setup the environment needed for the software development and evaluation of the Safe Driving Test cases. The relationships between hardware and software used for development are shown in Figure 3.1.

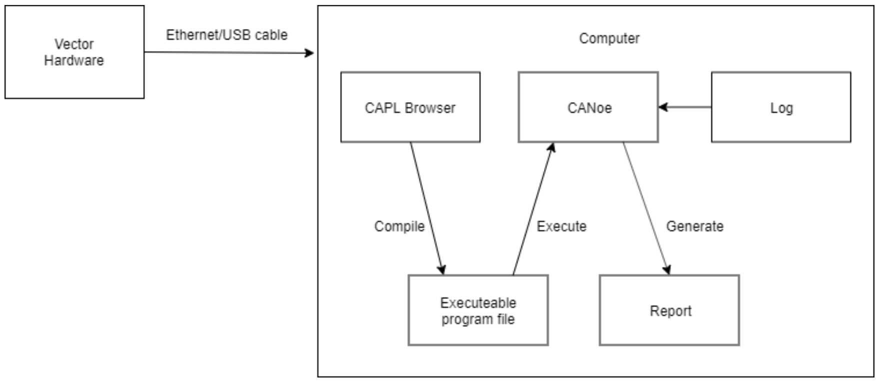


Figure 3.1: Overview of the environment setup.

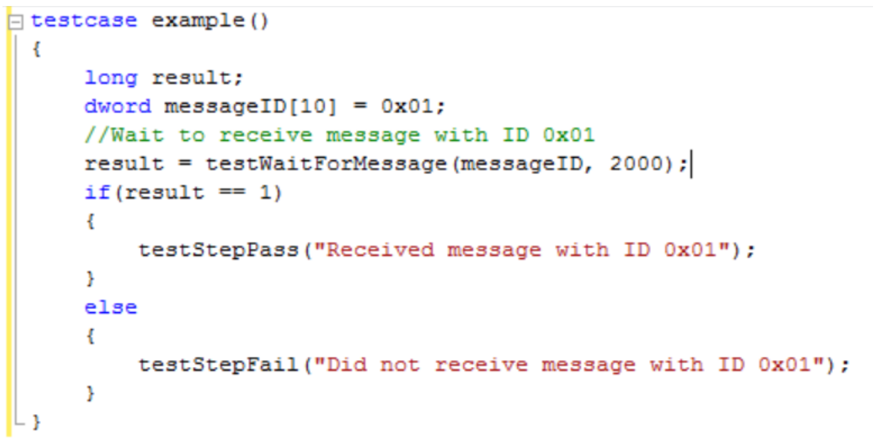
3.1 CANoe

CANoe is an application developed by Vector Informatik GmbH [Vector, 2019d] for development, test, analysis, simulation, diagnostics and start-up of the entire ECU networks and individual ECUs. The application is used in automotive in-vehicle electronic networking and also in industries such as heavy trucks, rail transportation etc. It is used worldwide as a tool for ECU development for all type of vehicles. The simulation and testing in CANoe is performed with CAPL, a programming language similar to C. The tool can be used to create a simulation of the entire network bus and simulate all the nodes on the buses. CANoe have support for all types of communication protocols used by automotive industry like CAN, CAN FD, LIN, FlexRay, Ethernet, MOST ect. In this project CANoe is used to be able to display the logged data, analyze the relevant signals and execute the implemented test cases.

3.2 vTESTstudio

vTESTstudio is a development environment developed by Vector Informatik GmbH [Vector, 2019e] for creating, structuring and generating automatic ECU test sequences which can be executed by CANoe configurations in real-time and evaluated in detailed reports. Tests are implemented in vTESTstudio by using test design languages such as CAPL or C#. In this master thesis vTESTstudio has been used for development of the evaluation program.

3.3 CAPL



```
testcase example()
{
    long result;
    dword messageID[10] = 0x01;
    //Wait to receive message with ID 0x01
    result = testWaitForMessage(messageID, 2000);
    if(result == 1)
    {
        testStepPass("Received message with ID 0x01");
    }
    else
    {
        testStepFail("Did not receive message with ID 0x01");
    }
}
```

Figure 3.2: A simple test case implemented in CAPL.

CAPL or CAN Access Programming Language is a procedural programming language based on the C programming language with additional features to support the development of the CAN-based embedded system. CAPL was developed by Vector Informatik GbmH [Vector, 2019a]. CAPL programs are developed and compiled using a dedicated program developed by Vector such as CAPL Browser or vTESTstudio. In this master thesis CAPL was used to implement the evaluation of test cases.

3.4 Log file

CANoe provides its user the ability to log all the data communication on the CAN buses. This function makes it possible for the user to analyze the record later after the tests have been performed. CANoe supports two type of logging formats: message-based and signal-based [Vector, 2019b]. The message-based format stores all the messages together with other information like statistics and disturbances on the buses. The signal-based format records all signal values extracted from messages sent over the network. Logging of the signal-based format requires a database for decoding signal values. In this master thesis the BLF (Binary Logging Format) file, a message-based format was used.

3.5 Vector Hardware

At BorgWarner, Vector Hardware is a crucial component for running software tests. To run the tests, the ECU is flashed with software must be connected to some of the hardware provided by Vector. The most common Vector hardware used at BorgWarner are VT-system and VN-system. They are slightly different but have the same purpose, to provide the ECU under test with a testing environment that is the same as in a real vehicle environment. Vector Hardware is also a necessary component to get a license for running CANoe.

3.6 Environment Setup

The environment needed for the software development and evaluation of the Safe Driving Test cases is setup using the equipment described above. Figure 3.1 shows the relationship between software and hardware. The Vector hardware is connected to the computer with CANoe via either Ethernet or USB. The purpose of the Vector hardware in this context is just to provide the license for using CANoe. It does not perform any simulation since this project only uses the real data traffic records from the log file. The software was written in CAPL with help of CAPL Browser. The CAPL Browser was chosen over vTESTstudio because both program provided similar functions but CAPL-Browser has less overhead.

4

Software Development

This chapter provides a detailed description of the development of the evaluation program.

4.1 Approach

Software development in general typically involves a number of common stages regardless of the methodology used. The six common main stages are [ElysiumAcademyPrivateLimited, 2017]:

1. Requirement elicitation
2. Software design
3. Implementation
4. Verification
5. Deployment
6. Maintenance

Based on the scope of this master thesis, the first four stages are relevant and were applied in this work. However, the stages were not followed strictly as in the waterfall model but was done iteratively. The presentation of this approach will nevertheless be organized after the order mentioned above. In the following sections, the stages will be gone through in detail and the result from each step will be provided.

4.2 Requirements

Before the design and implementation of the software begin, the requirements for the software must be elicited. It is of course difficult to gather all the requirements directly in this step but starting the software development with requirement elicitation is a good idea to get a deeper understanding of the problem the software will solve. The requirement elicitation methods used were: studying the current manual evaluation, interviewing software test engineers and studying the test specifications.

In order to be successful with this project I had two mentors at BorgWarner: Måns Andersson, supervisor and test verification engineer, who has the responsibility for creating and evaluating the SDTs and Mattias Wozniak, who is responsible for the software test architecture of all projects at the Software Testing Department.

At the initial phase I had several meetings with Måns. The purpose of the meetings was to provide me the necessary information about the problem I was going to solve. In these meetings he described how the manual evaluation of each test case was carried out and what should be included in the test reports. The description of the manual verification process is described in Section 2.2. This gave me an overview of the high-level expectations the Software Testing Department has on the proposed evaluation program.

During the development of the software I also had meetings with Mattias in which he went through and explained how the software test architecture of the current projects looks like and why it is designed that way. These meetings gave me the requirements on the software design. This also served as inspiration for me to come up with a design that fulfils these requirements.

The result of this step was a list of requirements on the software. Notable is that not all the requirements were formulated at this stage. Some of the requirements were created at the elicitation phase, but some have been added to the list during the course of this project. The reason why these requirements exist, how and what has been done to fulfil them will be explained in the following sections.

Below are the requirements of the software.

1. The evaluation program shall evaluate test cases from the Safe Driving Test based on logged data.

2. The output of the evaluation program shall be a test report containing test results for the evaluated test cases.

3. The evaluation program shall be easy to integrate into different projects.

4. The evaluation program shall be designed so that it is easy to expand it with new test cases.
5. Each project shall be able to declare its project specific limits without interfering with other projects.
6. Each log file will only contain data for one test case.
7. The evaluation program shall be able to deal with different signal names from different projects.
8. The evaluation program shall be able to evaluate software from different projects.
9. A test case is evaluated when the values of signals that are relevant to the test case pass the test condition check.
10. The signal data shall be sampled at minimum message cycle time which means every 10 milliseconds
11. Test condition check shall be created by translating the test instructions to a logical and programmable set of rules.
12. Depending on the expected test result the signal data shall be analysed in 3 ways: binary comparison, similarity comparison and value trend detection.
13. If the expected result of a test case is binary such as "value of signal A is greater than X" or "value of signal A is within an interval" and so on, the test result shall be analysed through simple binary comparison.
14. If the expected result of a test case is that the behavior of signal A is similar to the behavior of signal B, the test result shall be analysed by similarity comparison with three steps:
    - Step 1: Sample values from signal A and B in the evaluation interval.
    - Step 2: Time-align the signals using cross correlation coefficient.
    - Step 3: Evaluate the similarity between two signals using correlation coefficient. If signals has disturbances moving average is applied before using correlation coefficient.
15. If expected result of a test case is that "the values of signal A increase or decrease", the test result shall be analysed in either of the two ways below:
    - If the values of the signal shall strictly decrease or increase, binary comparison is applied which means that value of the incoming sample must be greater than value of previous sample, if expected result is increment and vice versa for decrement.

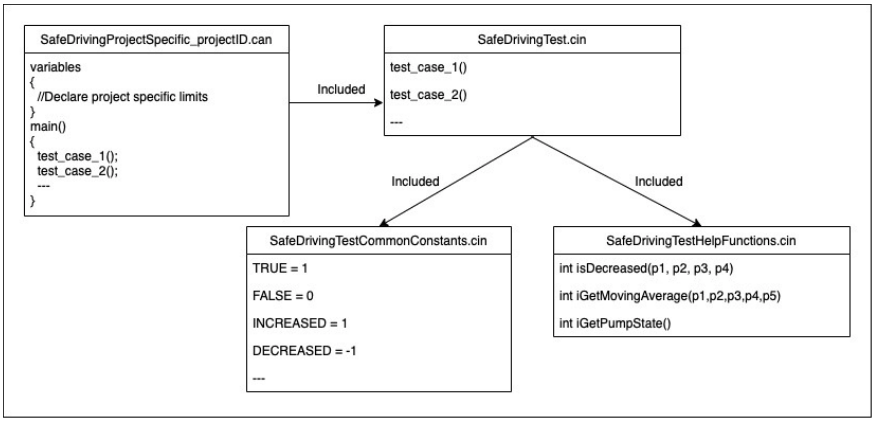
- If the values of the signal are not strictly decreased or increased which means the values can increase or decrease in some intervals but the overall trend is an increased trend or decreased trend, the evaluation shall be performed by first applying moving average on the signal and then using binary comparison to evaluate the increment or decrement.

4.3 Software Design

The structure of the software must be organized so that it is not only easy to understand, maintain and expand but also fulfills the requirements and needs. The requirements that have effect on the software structure are:

- Requirement 3: The evaluation program shall be easy to integrate into different projects.
- Requirement 4: The evaluation program shall be designed so that it is easy to expand it with new test cases.
- Requirement 5: Each project shall be able to declare its project specific limits without interfering with other projects.

The software structure presented in Figure 4.1 was created by taking all the requirements and needs into consideration.


SafeDrivingProjectSpecific_projectID.can
variables
{
//Declare project specific limits
}
main()
{
test_case_1();
test_case_2();

}
Included
SafeDrivingTest.cin
test_case_1()
test_case_2()

Included
Included
SafeDrivingTestCommonConstants.cin
TRUE = 1
FALSE = 0
INCREASED = 1
DECREASED = -1

SafeDrivingTestHelpFunctions.cin
int isDecreased(p1, p2, p3, p4)
int iGetMovingAverage(p1,p2,p3,p4,p5)
int iGetPumpState()

Figure 4.1: Overview of software structure when integrating into a project.

Requirement 3 and 5 are achieved by the "SafeDrivingProjectSpecific" file. This file has two purposes. The first one is declaring project specific limits by assigning the values to pre-defined global variables. This makes it possible for each project to declare its own limits. The second purpose is that it invokes only evaluation of test cases that are applicable for a specific project.

The SafeDrivingTest file contains the evaluation process implemented for all test cases in SDT. This file includes the SafeDrivingTestHelpFunctions file which is where all the algorithms and necessary functions supporting the evaluation process are implemented and the SafeDrivingTestCommonConstants, which contains all the common variables used by the SafeDrivingTest file. New test cases can easily be added to the evaluation program by implementing a new function for each test case in SafeDrivingTest file, and this fulfils Requirement 5.

4.4 Software Implementation

4.4.1 Log File Preprocessing

Requirement 6 limits the number of test cases that can be logged in a log file. The reason why this requirement exists and how this issue was solved is explained in this section.

After examining the log files for each test case from a former SDT, it turned out that one log file can sometimes contain logged data for multiple test cases. This is due to the overlapping test sequences between these test cases. Two test cases might have almost identical driving scenario but with a small difference. For example, both test cases can be: accelerate to 20 km/h and then perform a turn. The difference here is that in one test case there will be a left turn and in the other a right turn. An example of overlapping sequences can be: test case 1, start the engine and test case 2, start the engine and wait for the engine to idle. Test case 2 overlaps test case 1. The overlap sequence here is of course the "start engine"-step.

For the sake of simplicity, the test engineer often performs and records these test cases together in one log file. However, this manner makes the implementation of the evaluation for each test case complicated and unreliable. The evaluation must somehow be able to detect and distinguish between test cases to be able to determine when to start evaluating a specific test case. For example, a log file may contain log data for test cases 1, 2 and 3. The evaluation program needs to know which part of the log file belongs to each test. To ease the implementation and therefore minimize the risk of evaluating the wrong test case, I decided that if a log file contains data from more than one test case it must be preprocessed before usage.

Preprocessing in this context means that a log file, which contains data from more than one test case, must be split up into separate files. CANoe at this moment does not offer a tool for dividing a log file. However, CANanalyzer has a tool for that.

The user could load the file he wants to split and enter a start timestamp and an end timestamp of a test case in the log. The result is a file that only contains log data for the specified interval.

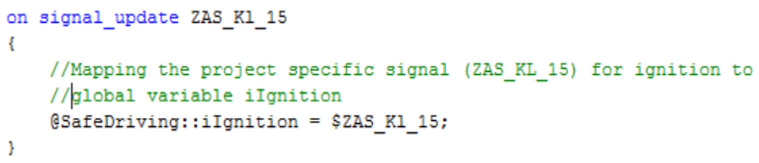
Preprocessing of log files cannot be automated so the user must do it manually. It requires that the user can differentiate between test cases by looking at relevant signals or by saving the test case separately during testing.

4.4.2 Generic Solution for Customer-Specific Configurations

BorgWarner provides products for numerous customers, i.e., various car manufacturers. Even though it is the same product, each car manufacturer has its own configuration and therefore has different requirements on how the product should communicate with the rest of their system, the behavior of the software at certain situations, which signal should show which data, customized features, etc. To make it as effective as possible the testing department at BorgWarner requires that the evaluation program must be designed and implemented so that it can be used across multiple projects. In this section I am going to address the customer specific problems and describe my solution to make the evaluation program generic.

Project-specific Signal Names
The evaluation of a test case is performed by analyzing the values or value trend of relevant signals. For each test case there is a group of signals that are of interest, and it happens to be that the names of the signals are not the same across different customers. One signal can have the same purpose but different name depending on the car manufacturer. For example, the name of the wheel speed signal could be *wheelspeed* for car manufacturer A but could be *wheelspd* for car manufacturer B. The variation in signal names between projects prevents the program from being reused for various projects. A solution for this problem must also satisfy Requirements 3, 7 and 8.

To solve this problem, I studied how the earlier implementation of other test modules solved the customer specific problem. The solution was that each project using the evaluation program must map its signals to the predefined global and common variables. I summarized a list of signals used in the evaluation of all test cases and created global variables for these signals. Each project will have to implement the mapping between its project specific signals and the global variables. This mapping updates the variables with the value of the project specific signals as soon as any change in value from these signals occurs. This way the evaluation program can analyze data from the global variables instead of dealing with the project specific signals directly. Figure 4.2 shows an example of how the mapping is done in CAPL.



```
on signal_update ZAS_Kl_15
{
    //Mapping the project specific signal (ZAS_KL_15) for ignition to
    //global variable iIgnition
    @SafeDriving::iIgnition = $ZAS_Kl_15;
}
```

Figure 4.2: Signal mapping.

Different Signal Units and Project-specific Limits
BorgWarner's customers also use different units for their measured data. For example, in some projects the wheel speed can be measured in meters per second, while in other it is measured in kilometers per second. This problem is solved by specifying a unit for each signal that is used by the evaluation program. A signal with a unit that differs from the one specified in the evaluation program has to be converted to the specified unit. That way the evaluation will be generic across projects.

Another problem is that the limits are not the same across projects. For instance, in Project A the pump is off when the ampere is equal or less than 50 mA, while in Project B the pump is off when the ampere is equal or less than 30 mA. If these limits are not taken into consideration the evaluation program will provide incorrect test results.

To give an idea, one of the test cases in SDT is to turn off the motor and check that the pump is off. The test case is performed on Project A and B, and the ampere values received are 40 mA respective 30 mA, which means the pump was off in both projects. The test passed in manual evaluation. Now assume that the limit for pump being off is hardcoded to 30 mA in the automatic evaluation program. The automatic evaluation program will evaluate the test result in Project A as fail since the ampere is greater than 30 mA, while it will evaluate the test result from Project B as pass. In this case the evaluation program gives incorrect evaluation for Project A.

To solve this problem I decided that limits that are different from project to project will have a global variable and this variable has to be assigned a value by each project using the evaluation program.

4.4.3 Algorithms

The automatic evaluation workflow is shown in Figure 4.3. The two most important parts of the automatic evaluation is the test condition check and the evaluation. The test condition check is responsible for detecting intervals where a test case is performed in the log. This is necessary because a log file does not only contain data for just the duration the test case is executed but it also contains data for the period before, between and after the test execution in which some intervals are not related to the test. Figure 4.4 shows log of a test case. The same test case was performed two times (intervals within blue rectangles). The intervals before, between and after the test execution (within magenta rectangles) are not related to the test case and should not be evaluated. A test case might be repeated multiple times; therefore, it is absolutely essential that the evaluation process must be able to identify the start and the end of the test case and be able to find all the intervals where a test case is executed using the provided data. The program should only evaluate intervals where a test case actually occurs (Requirement 9). The input of the test condition

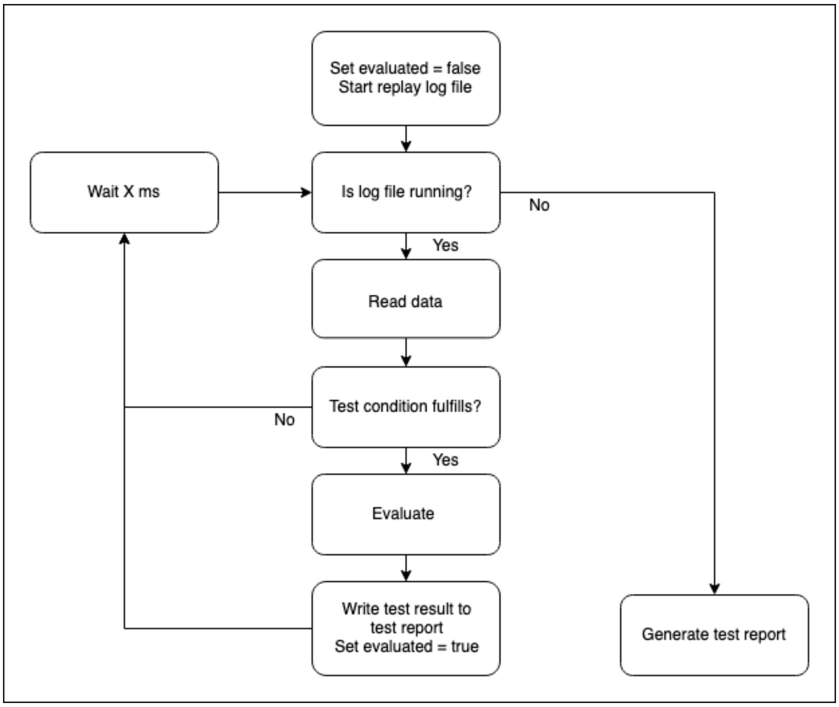


Figure 4.3: Automatic evaluation workflow.

check are the values of relevant signals. Since the messages contain the value of different signals are send in different frequency depending on the message cycle, I decided that the signal data should be sampled every 10 milliseconds which is the minimum message cycle. The reason is to make sure that no important sample values are missed (Requirement 10).

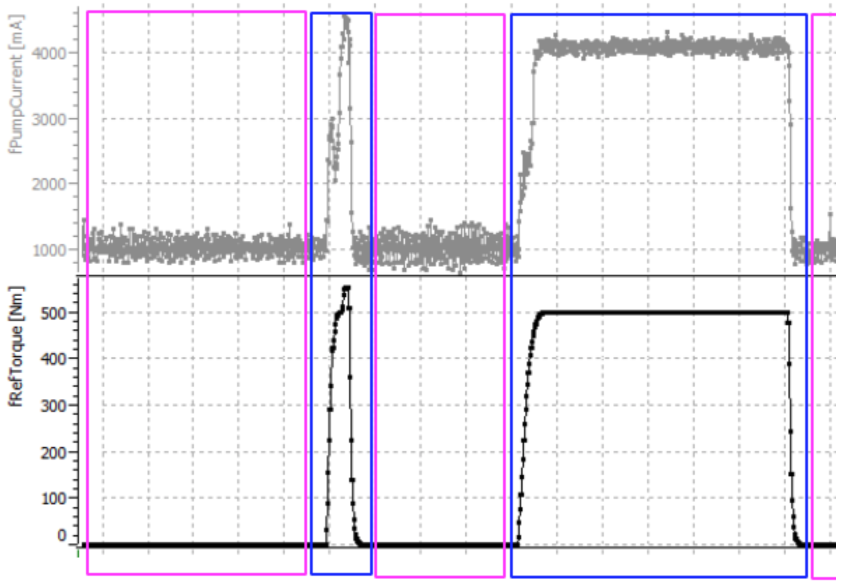


Figure 4.4: Intervals within blue rectangles are where the test case is executed. Intervals within magenta rectangles are not related to the test case and therefore should not be evaluated.

Figure 4.7 provides an example of how the program shows which intervals it evaluates. Each test case has a description of the driving scenario, so the test condition check was implemented by transforming the description into a set of programmable rules (Requirement 11). For visualization of the evaluation intervals, there is a signal called "iVerifying" which is the orange signal in Figure 4.7. The evaluation intervals are the intervals in which the signal "iVerifying" is 1. The iVerifying is 1 when the test condition check is fulfilled.

After these intervals are detected the evaluation program proceeds to the next step: evaluation. Depending on the expected result of the test case the evaluation can be divided into two groups: binary evaluation and complex evaluation (Requirement 12). The binary evaluation is as simple as comparing the result value with the expected value, while the complex evaluation means to verify that the value of a signal changes over time in an expected pattern. For instance, the ampere of the pump is expected to increase when the accelerator pedal is pressed and decrease when the accelerator pedal is released.

Depending on the expected result of each test case one or a combination of different evaluation approaches are applied. In the following sub-section, the algorithms or methods implemented in CAPL for determining the evaluation intervals and evaluation of a test case along with the motivation for the chosen algorithms will be presented.

Wheel Slip Detection
In the SDT there are several test cases in which are designed to test how the software handle situations where one or both of the front wheels lose traction, while turning left or right. In order to evaluate these test cases, the evaluation program must be able to identify intervals where the wheels lose grip based on the available signal data.

In a real driving situation, when the wheel slip occurs the driver should easily be able to detect that by hearing the sound from the wheels and by feeling the slipping when steering. It is also easy for a human to see where the wheel slip happens only by looking at the displayed logged data. Figure 4.5 shows data of the front wheel speeds. We can see that at normal driving with no wheel slip the signal data for both wheels look smooth. When wheel slip occurs on the right wheel, we could see that the signal oscillates much more compare to the signal of the left wheel.

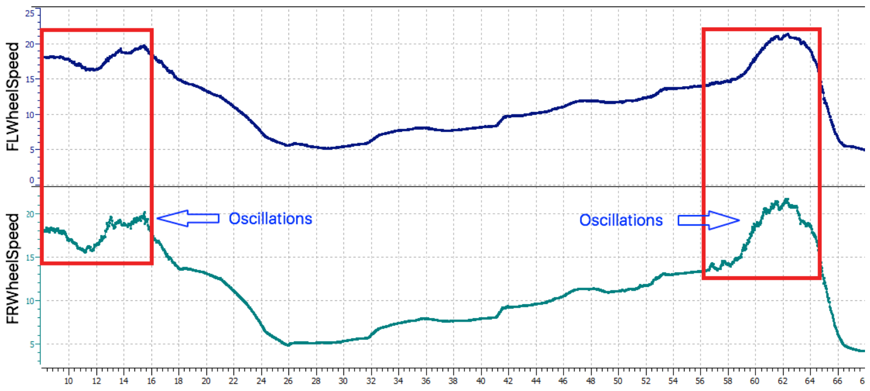


Figure 4.5: Intervals where wheel slip occurs. Front left wheel speed in blue and front right wheel speed in turquoise.

Nevertheless, to make the computer sees what we see and comes to the same conclusion is not easy. To implement wheel slip detection based on available data requires some knowledge about physics. As a starting point the behavior of the front wheels when turning left or right was studied. In a normal driving situation without wheel slip when turning left, the right wheel must travel a longer distance compared to the left wheel which means that the right wheel should always has a higher speed than the left wheel. The same principle is applied to a right turn, except that the left wheel will always has higher speed than the right wheel. Figure 4.6 shows the technical details of the wheels when cornering.

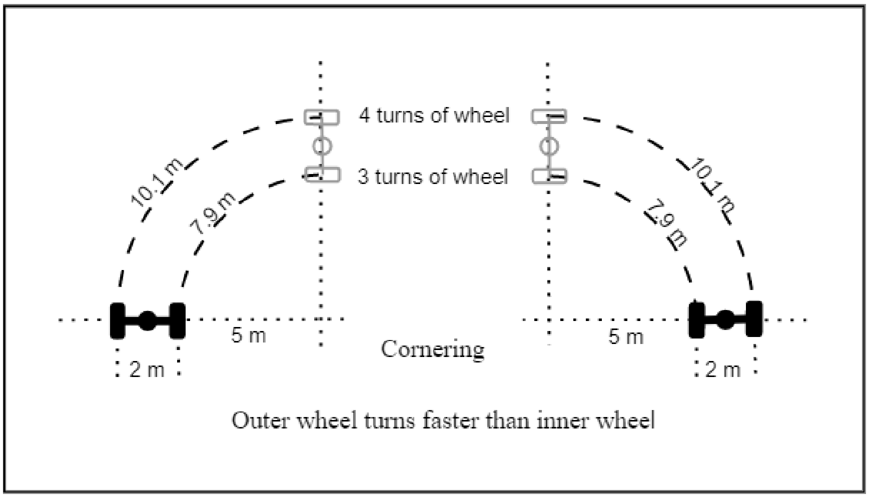


Figure 4.6: Cornering

Ideally, for detecting wheel slip when cornering one could calculate the expected wheel speed for each wheel and compare with the actual wheel speed. The expected wheel speed can be calculated based on the traveled distance of each wheel, the turning angle, the distance between front wheels and the time it takes for the wheels to travel that distance. Unfortunately, the log data does not contains information for any of the mentioned parameters. The information that can be extracted from logged data is wheel speed for each wheel, steering wheel angle and the steering wheel direction. Therefore, the evaluation program can only use the available data for detection. By combining the knowledge about the wheel speed when cornering and studying the logged data of wheel slip test cases a programmable set of rules for detecting a wheel slip situation can be created.

At a left turn wheel slip occurs when:

- The steering wheel angle is greater than for example *x* degrees.
- The steering direction is left.
- The left wheel speed is greater than or equal the right wheel speed

The reason why the first rule exists is because when the cornering angle is small the speed difference between left and right wheel is almost negligible. Since the speed difference is negligible when steering wheel angle is small, it is impossible to detect if wheel slip occurs or not using the rules mentioned above. If we can calculate the expected wheel speed it is then possible to detect wheel slip even at small steering angle. The second rule decides which wheel speed will be the reference. This rule is a complement to the third rule. For instance, when turning left the right wheel speed is the reference. If the left wheel spins faster than or equal to the reference wheel, wheel slip occurs.

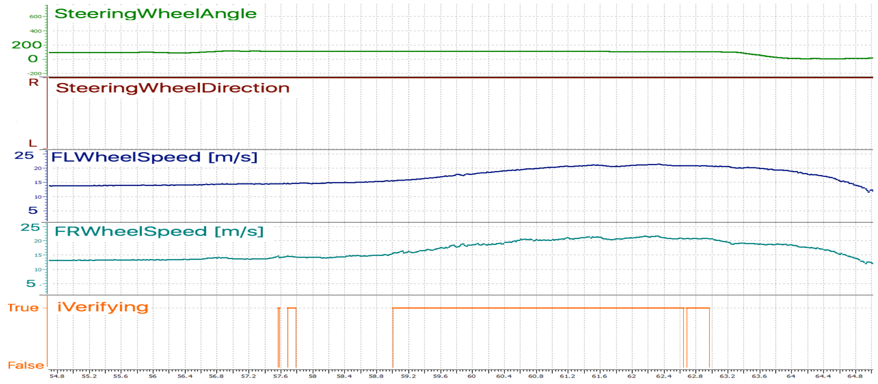


Figure 4.7: Wheel slip detection at right cornering. Signals in separate diagrams.

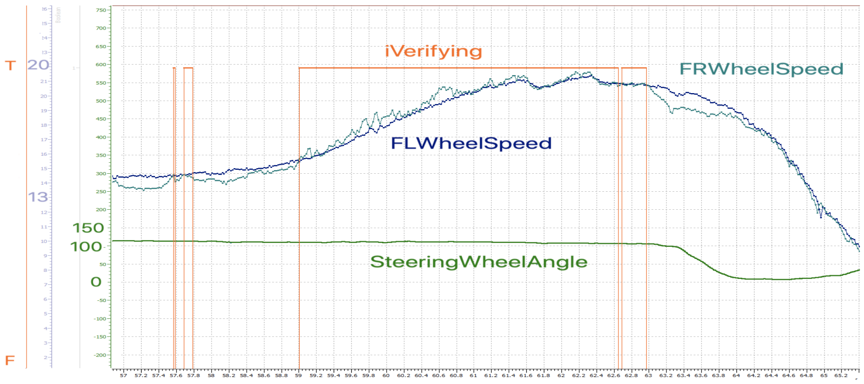


Figure 4.8: Wheel slip detection at right cornering. All signals with the same y-axis.

Figure 4.7 and 4.8 show the result of the wheel slip detection. In Figure 4.7 when showing the signals in separate diagrams it is difficult to see where exactly the wheel slip occurs but in Figure 4.8 when placing both wheel speeds in the same axis the wheel slip occurrence is clearer. The intervals within the orange signal is where the wheel slip occurs. The interval on the right of Figure 4.8 looks like wheel slip also occurs but the wheel slip actually did not happen in that interval. As mentioned above when the steering angle is small the wheel speed is almost the same for both wheels and because of the first rule that interval is not detected as wheel slip.

Accelerator Pedal Lift Detection
Some test cases are supposed to test that an expected feature is applied when the driver releases the accelerator pedal. Since this feature expects to be applied as soon as the acceleration pedal is released, the evaluation program must be able to detect where in the log the accelerator pedal is released. The information available from the accelerator pedal is the accelerator pedal position in percent between 0 to 100. Detecting the accelerator pedal lift might be as easy as finding the points when the accelerator pedal position goes to zero. However, this assumption is not true because as one could imagine when the driver removes his foot from the pedal, the pedal does not go back to the zero position immediately, but it takes some milliseconds for the pedal to reach the zero position. This means that the feature expects to be applied somewhere between these milliseconds.

Studying the signal of the accelerator pedal position reveals that when the pedal is released the decrement of the accelerator pedal in percent is much faster than normal. Due to this fact the accelerator pedal lift detection is implemented using the gradient and its direction. A negative gradient means the accelerator pedal position is decreasing.

The accelerator pedal lift is detected when it has a high negative gradient. The gradient is given:

$$g(percent/second) = \frac{x[i+1]-x[i]}{t[i+1]-t[i]} \tag{4.1}$$

where x is accelerator pedal position signal and t is the time stamp corresponding to the sample.

Moving Average
The correlation coefficient is used in signal comparison. The correlation coefficient performs really well on signals without disturbances but not so well on signal with disturbances. Consequently, to make the comparison more accurate, signals with disturbances must be filtered before using as input to the comparison. For that purpose, the moving average filter is used [Steven W. Smith, 1999]. Moving average is a simple filter but it is very effective for making the shape of the signal stand out (Requirement 14 Step 3). The moving average filter works by shifting a window with fixed size one point at a time over the input signal while computing the mean value of the points fit into the window size. The mean value of each shift becomes a sample of the output signal.

The equation is given by:

$$y[i] = \frac{1}{n}\sum_{j=0}^{n-1} x[i+j] \tag{4.2}$$

where n is the window size, x is input signal and y is output signal. The evaluation program will first sample the signal to be filtered over the entire evaluation period and then apply the moving average filter to the signal.

Delay Determination Between Signals
A vehicle is built from several systems like engine system, steering system, braking system, etc. Each of these systems are controlled by ECUs. In today's modern vehicles there are hundreds of ECUs installed. To operate properly and effectively the ECUs communicate and exchange information with each other by sending messages containing data over the CAN bus.

Even though the communication over the CAN bus and the time it takes for an ECU to receive input from sensors and switches is really fast, there are still delays between signals. It takes a small amount of time for the message to be transmitted from ECU A to ECU B and a small amount of time for a signal to travel from switches and sensors to ECU. Therefore, when an event occurs it will take some time for the ECU to get that information and reply with a command.

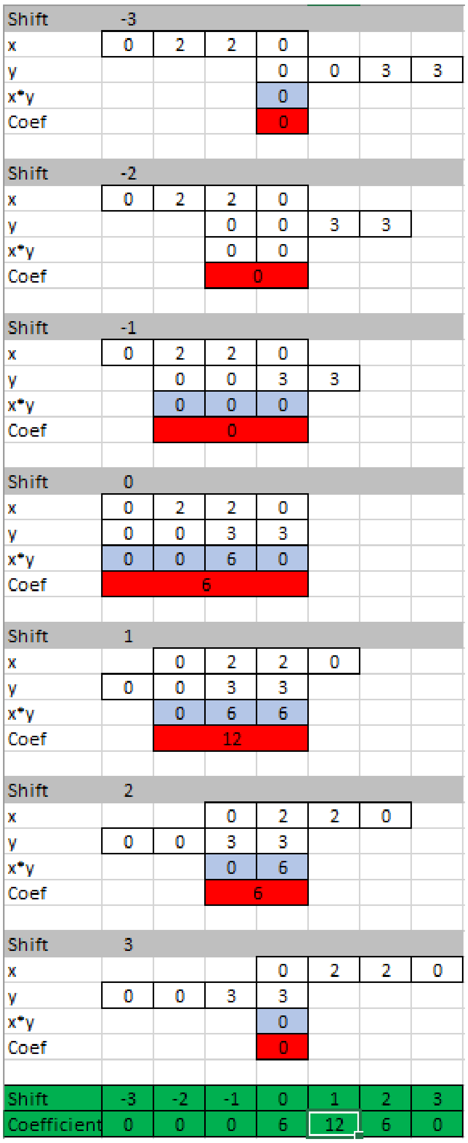
Since there is almost always a delay between signals, evaluation of the test cases where an event occurs and expects some specific reaction from the software to be activated, it is important to find the delays between relevant signals. By finding the delay, evaluation programs can evaluate if the reaction time is within the expected time interval. Besides that, identifying the delay is also important in signal comparison which will described in the next section.

For determining the delays between signals, a method called cross-correlation is used (Requirement 14 Step 2). Cross-correlation is a very useful method with many practical applications in signal processing and one of them is identifying delays. Delay determination is done by finding the cross-correlation of two signals. The cross-correlation can be calculated by computing the sum of the product of the overlapping samples while sliding a signal upon the other signal. The cross-correlation of discrete time signals is given by:

$$C_{xy}[j] = \sum_{i=0}^{n} x[i] * y[i-j] \tag{4.3}$$

Where $C_{xy}[j]$ is correlation coefficient of signal x and y at sample j. n is the number of samples. j is a value between $-n$ and n. i is a value between 0 and n.

For instance, the cross-correlation of signals $x[i] = [0,2,2,0]$ and $y[i] = [0,0,3,3]$ can be calculated as shown in Figure 4.9. In Figure 4.9 the samples with blue color



Shift	-3						
x	0	2	2	0			
y				0	0	3	3
x*y				0			
Coef				0			
Shift	-2						
x	0	2	2	0			
y			0	0	3	3	
x*y			0	0			
Coef			0				
Shift	-1						
x	0	2	2	0			
y		0	0	3	3		
x*y		0	0	0			
Coef		0					
Shift	0						
x	0	2	2	0			
y	0	0	3	3			
x*y	0	0	6	0			
Coef	6						
Shift	1						
x		0	2	2	0		
y	0	0	3	3			
x*y		0	6	6			
Coef		12					
Shift	2						
x			0	2	2	0	
y	0	0	3	3			
x*y			0	6			
Coef			6				
Shift	3						
x				0	2	2	0
y	0	0	3	3			
x*y				0			
Coef				0			
Shift	-3	-2	-1	0	1	2	3
Coefficient	0	0	0	6	12	6	0

Figure 4.9: Cross correlation calculation

are obtained by multiplying the corresponding sample of *x* with the corresponding sample of *y*. The samples with red color are the correlation coefficient when shifting a signal *x* positions to the left or right from the other signal. These sample are obtained by adding the samples with blue color.

The cross-correlation obtained from the signals *x* and *y* is [0, 6, 12, 6, 0, 0, 0] where the fourth element is the zeroth sample which means there are no shifting between *x* and *y*. To the left of the zeroth sample is the coefficient of two signals when signal *x* is shifted to the left relative to signal *y*. To the right of the zeroth sample is the coefficient of two signals when signal *x* is shifted to the right relative to signal *y*.

Further, the cross correlation is at its highest value, 12 which means that the two signals are most similar when signal *x* is shifted one position to the right. This also means that the delay between signal *x* and *y* is 1 sample. Figure 4.10 shows signal *x*, *y* and the shifted version of *y*. With bare eyes we can see that signal *y* is delayed one sample relative to signal *x*. Using the Matlab function for computing the cross-correlation we get the same result. It is worth noting that the cross-correlation can still find the correct delay even though the amplitudes of these signals are different.

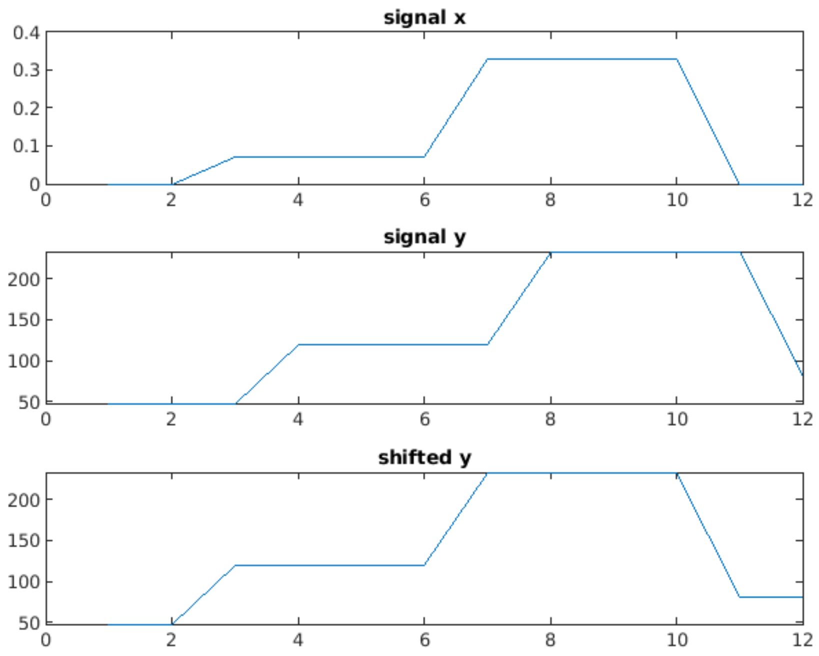


Figure 4.10: Using cross-correlation to find the delay between signal *x* and *y*

Similarity Comparison Between Signals

Test cases with expected result such as "signal B will increase when signal A increases and decrease when signal A decreases" are very simple to evaluate manually by optical comparison. Figure 4.11 shows an example of two signals. The first one is the ampere signal and the second on is the torque signal. The ampere signal is expected to mirror the torque signal. For the human eyes there is no problem to see that the ampere signal behaves similar to the torque signal even though the ampere signal has oscillations in it. For the automatic evaluation to achieve the same conclusion as the manual evaluation an appropriate algorithm for comparing the similarity between two signals is needed. The algorithm must also be able to compare signals with different amplitude, signals with disturbances and signals with delays.

After studying different algorithms that are used in machine learning and signal processing for signal/data comparison, an algorithm that is suitable for solving this problem was found. The algorithm is called correlation coefficient.

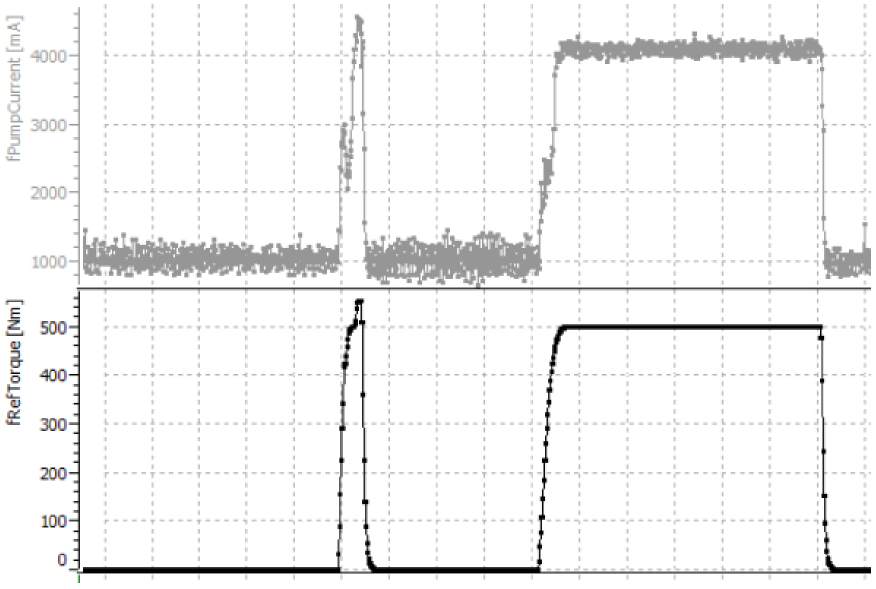


Figure 4.11: Ampere signal and torque signal.

In statistics the correlation coefficient is used to measure how strong the relationship is between the value trend of two variables. The correlation coefficient is a number between -1 and 1. -1 is the strongest negative correlation and 1 is the strongest positive correlation. A negative correlation coefficient means that if one variable increases the other variable decreases while a positive coefficient means that if one variable increases the other variable also increases. The closer the absolute correlation coefficient is to 1 the higher the correlation is between signals or in other word the more similar two signals are. The formulation used to calculate the correlation coefficient is given below. Assume the data of signal x and y stored in array x and y. The correlation c is:

$$c = \frac{\sum_{i=0}^{n} (z_x)_i (z_y)_i}{n-1} \tag{4.4}$$

$$(z_x)_i = \frac{x_i - \overline{x}}{s_x} \tag{4.5}$$

$$(z_y)_i = \frac{y_i - \overline{y}}{s_y} \tag{4.6}$$

$$s_x = \sqrt{\frac{\sum_{i=0}^{n} (x_i - \overline{x})^2}{n-1}} \tag{4.7}$$

$$s_y = \sqrt{\frac{\sum_{i=0}^{n} (y_i - \overline{y})^2}{n-1}} \tag{4.8}$$

Where $(z_x)_i$ and $(z_y)_i$ are the standardized value of signal x and signal y at index i of the data arrays, s_x and s_y are the standard deviation of signal x and y, $\overline{x}$ and $\overline{y}$ are the mean value of signal x and y and n is the number of samples.

As mentioned earlier the goal is to find an algorithm that manages to compare signals with disturbances, delays and different amplitudes. The correlation coefficient method, however, only manages to compare signals with different amplitude. Since I did not find an algorithm that could solve this completely, I combined several different algorithms to achieve the goal.

For signals with some delay but no disturbance, the delay between them must first be found using the cross-correlation described in Section 4.4.3, after that the program adjust the signals by shifting one signal to the left or right relative to the other signal based on the delay so that they are time-aligned and finally the time-aligned signals are used as input to the correlation coefficient algorithm.

For signals with both delays and disturbances, the signals must first be shifted according to the delay, then filtered or smoothed using the moving average algorithm and after that the resulting signal can be used as input to the correlation coefficient.

5

Verification and Result

This chapter provides a description of the verification process as well as discussion and analysis of the result obtained from the verification. The verification was done by integrating the evaluation program to a project and running the evaluation program on the log files from a former SDT of that project. The report provided by the evaluation program was then analysed thoroughly based on two factors: detection of the evaluation interval and the evaluation result of these intervals. The chapter ends with an example how a specific test case is evaluated by the program.

5.1 Verification

In this stage the software was tested to verify that it meets the specifications and works as intended. The evaluation program was integrated into a project and the logs from a former SDT of that project were used as input to the evaluation program. The test report generated from the evaluation program was then analysed exhaustively and compared to the manual test report, test case by test case.

The analysis was done based on two factors which are important for the reliability of the evaluation program: detection of the evaluation interval and the evaluation result of these intervals. The detection of an evaluation interval is accurate if it can find all evaluation intervals that match the test scenario of the specific test case. The evaluation is accurate when it gives pass result when the test result is as expected and fail when it is not as expected. If the two factors are accurate the evaluation program is reliable. The result from the analysis and comments are categorized in two groups and presented in the subsections below.

5.1.1 Evaluation Interval Detection

There are two ways to verify if the detection of the evaluation interval is correct or not. The first one is to compare the intervals detected by the program with corresponding description of the test scenario from the test instruction. This method is applied for test cases that have simple test scenarios such as "while driving at 20 km per hour engage brake". The verification is performed by checking if the wheel speed is 20 km/m when the brake signal switched from 0 (inactive) to 1(active) in the detected intervals. Besides that, I also verify that all intervals that matches the test instruction are detected. The second one is to discuss with domain expert. When verifying test cases which may be ambiguous to verify if the detected intervals are correct or not I often asked for help from test evaluation engineer to make sure that the verification was correct.

The analysis on the detection of the evaluation intervals showed that the evaluation program detected the correct intervals in 14 of 18 test cases and incorrectly in 4 test cases. In Figures 5.1, 5.2, 5.3 and 5.4 the intervals that have been incorrectly detected are the intervals within the red rectangle box. These four test cases have similar driving scenarios but are slightly different when it comes to steering direction and accelerator pedal. The driving scenario is accelerating the vehicle and then performing a minimized right respective left turn while the accelerator pedal is pressed respective released. Since the driving scenarios are similar the implementation of the detection for them are also similar, and therefore the same defects occur in these test cases.

The reason for these defects was that the detection can only see the value of these signals at the current moment. As soon as values of the signals pass the test condition check, it concluded that the test case is happening, and the evaluation flag is set to 1. While the evaluation flag is 1, the evaluation program keep evaluating as long as the incoming samples pass the condition check. The evaluation program therefore does not know that the test case was not performed in these intervals but it was just some sample values that happen to pass the test condition check. Figure 5.1 shows two evaluation intervals. The evaluation intervals are intervals where signal "iVerifying" is true. The first evaluation interval was wrongly detected because there are samples that passed the test condition check and the evaluation flag is set to 1 but the test case was not performed at that time.

In addition to correct detection in 14 of 18 test cases the analysis also reveals that the evaluation program is able to detect intervals that are likely to be missed in manual evaluation, especially in test cases about wheel slip. Figure 5.5 and 5.6 show the evaluation intervals detected by the evaluation program. Within the green rectangle boxes are intervals which are hard to detect optically. Based on the behaviors of the signals in Figure 5.5 it is really difficult for the human eye to see in which intervals the wheel slip appears. Besides intervals that can be found optically the evaluation program also finds intervals which are hard to find with the human eye.

After the analysis it can be concluded that in general the evaluation program can find all the intervals it should evaluate but in some test cases there are intervals that should not be included in the evaluation.

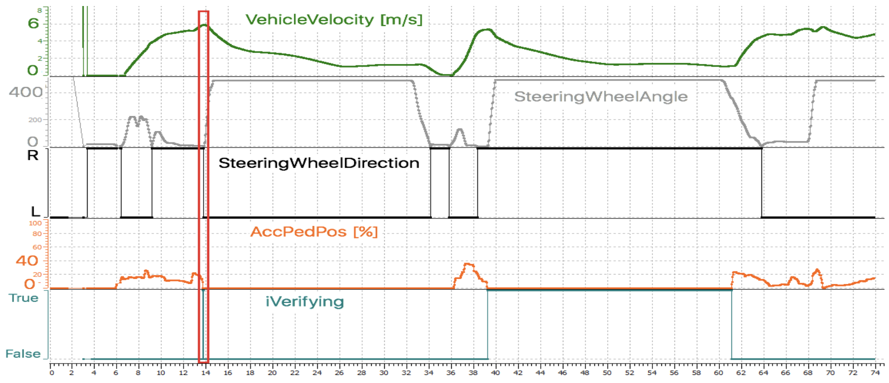


Figure 5.1: Evaluation detection of Test Case 9. The interval within the red rectangle was wrongly detected.

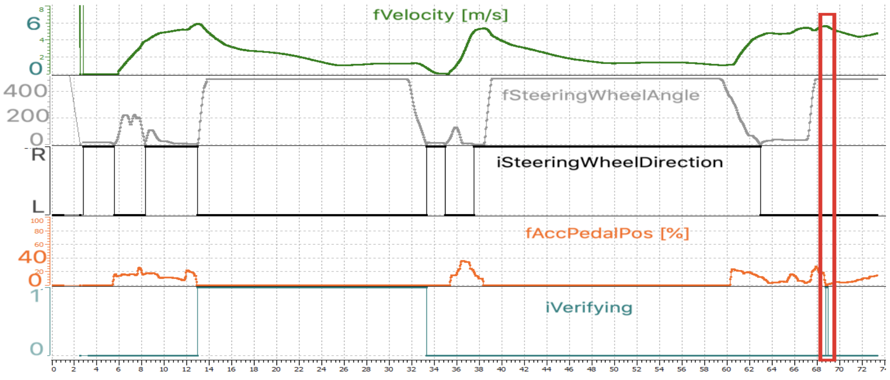


Figure 5.2: Evaluation detection of Test Case 10. The interval within the red rectangle was wrongly detected.

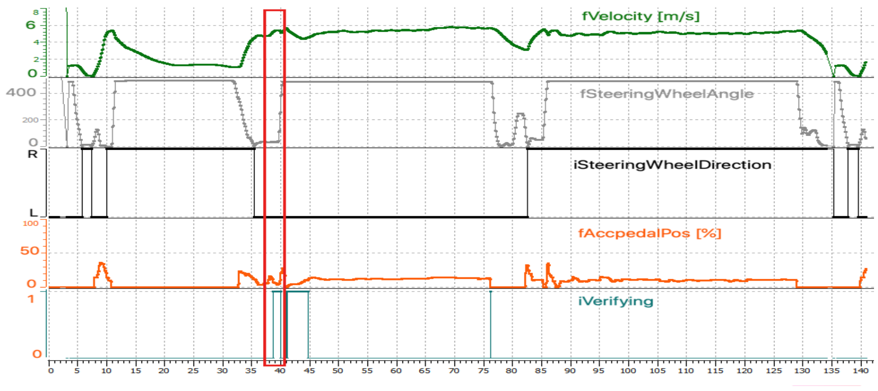


Figure 5.3: Evaluation detection of Test Case 11. The interval within the red rectangle was wrongly detected.

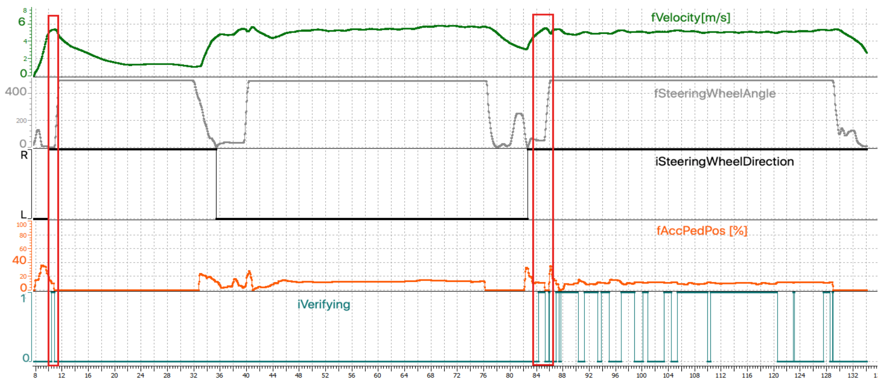


Figure 5.4: Evaluation detection of Test Case 12. The interval within the red rectangle was wrongly detected.

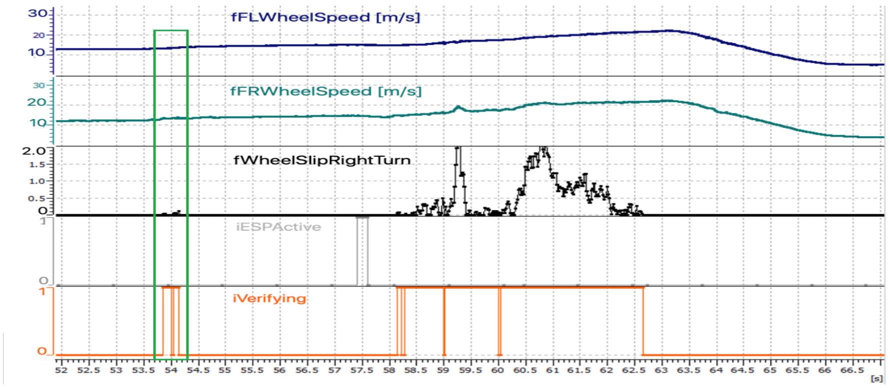


Figure 5.5: Evaluation detection of Test Case 15. The interval within the green rectangle was detected by the evaluation program but can be missed in manual evaluation.

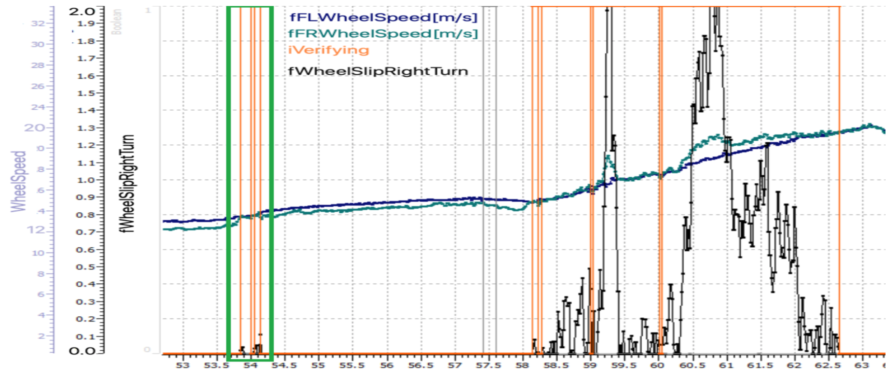


Figure 5.6: Evaluation detection of Test Case 15. The interval within the green rectangle was detected by the evaluation program but can be missed in manual evaluation.

5.1.2 Evaluation Process

To verify the evaluation process of the software the evaluation result was analysed and compared with the manual evaluation. In the manual test report the number of test cases passed are 16 of 18 test cases. However, only 9 of 18 test cases passed the automatic evaluation. The analysis of the passing test cases showed that the evaluation was correct, and therefore, only the analysis of failing test cases will be mentioned here.

There are three reasons behind the differences in test result between the manual and automatic evaluation. The first reason is incorrect evaluation intervals in the automatic evaluation. As mentioned in the previous subsection in test cases 9, 10, 11, 12 there are intervals that are detected by the evaluation program but should not be included in the evaluation. These test cases should pass the evaluation but due to incorrect evaluation intervals it failed.

The second reason is that the test case actually failed but was not detected in the manual evaluation. This happened in Test Case 17 and 18. These test cases tested that the torque increased when the accelerator pedal was lifted. In Test Case 17 there was an interval where the torque did not increase but this was probably not detected in the manual evaluation. In Test Case 18 by observing the behavior of the torque signal it can be seen that when the accelerator pedal signal goes to zero the torque increases as expected, but actually the torque increased much later. This failure was probably not seen in manual evaluation. The behavior of the torque signal in Figure 5.9 seems to be as expected, but the zoom in of these signal in Figure 5.10 reveals that the test failed in the last three trials.

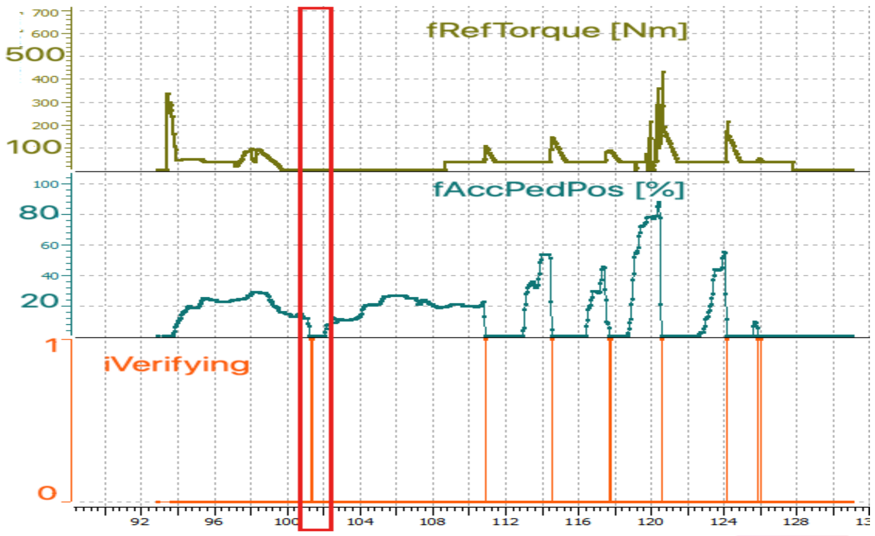


Figure 5.7: Interval was missed in manual evaluation of Test Case 17. The interval within the red rectangle was probably not evaluated in manual evaluation.

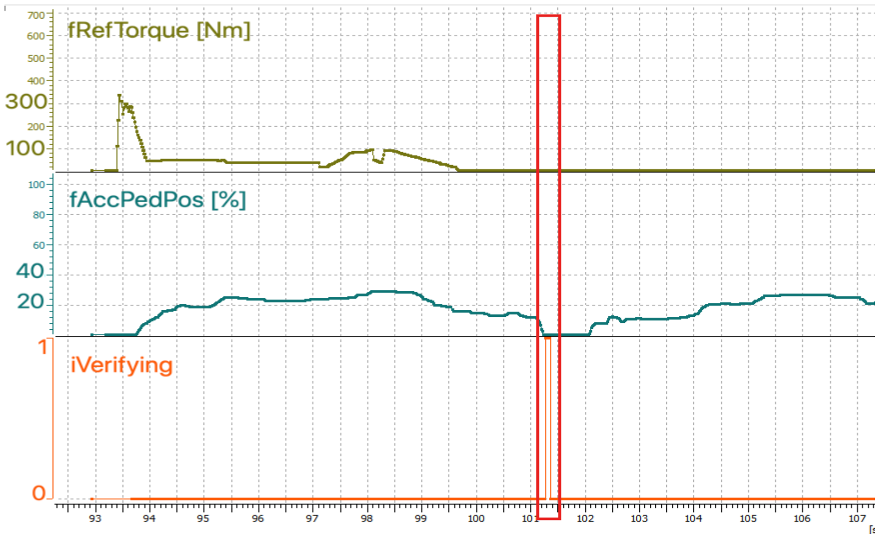


Figure 5.8: Interval was missed in manual evaluation of Test Case 17 (zoom in on Figure 5.7). The interval within the red rectangle was probably not evaluated in manual evaluation.

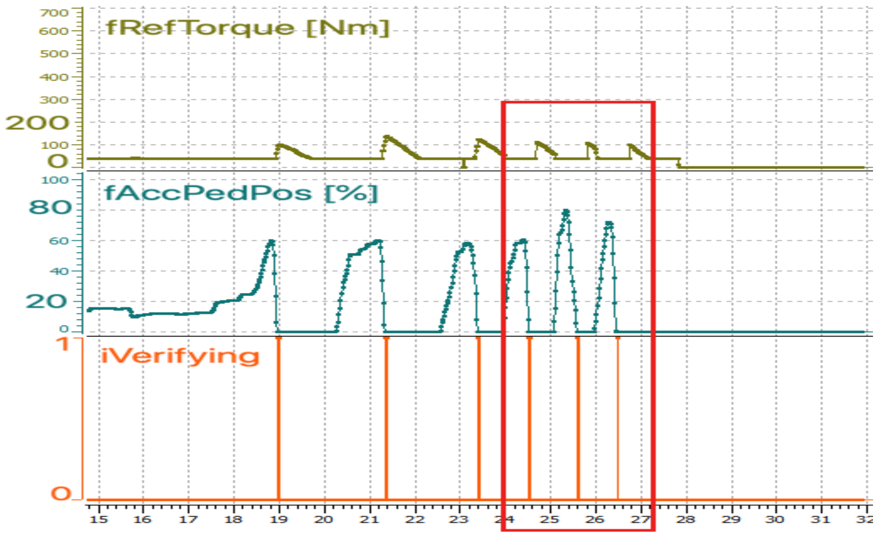


Figure 5.9: Test Case 18 passed in manual evaluation but actually failed. The interval within the red rectangle was probably wrongly evaluated in manual evaluation.

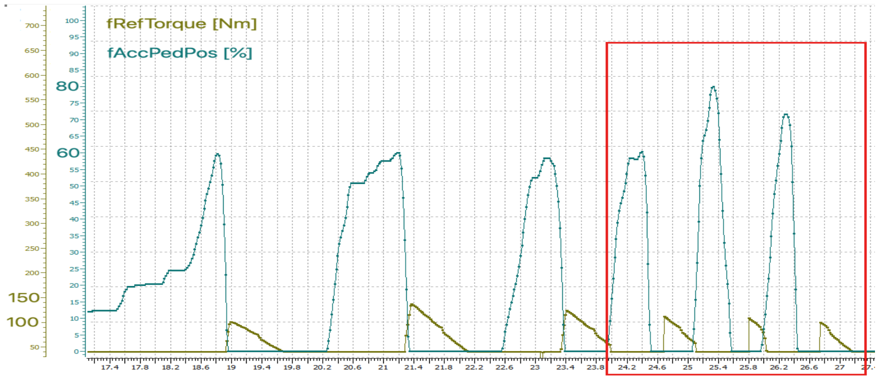


Figure 5.10: Test Case 18 passed in manual evaluation but actually failed (zoom in on Figure 5.9). The interval within the red rectangle was probably wrongly evaluated in manual evaluation

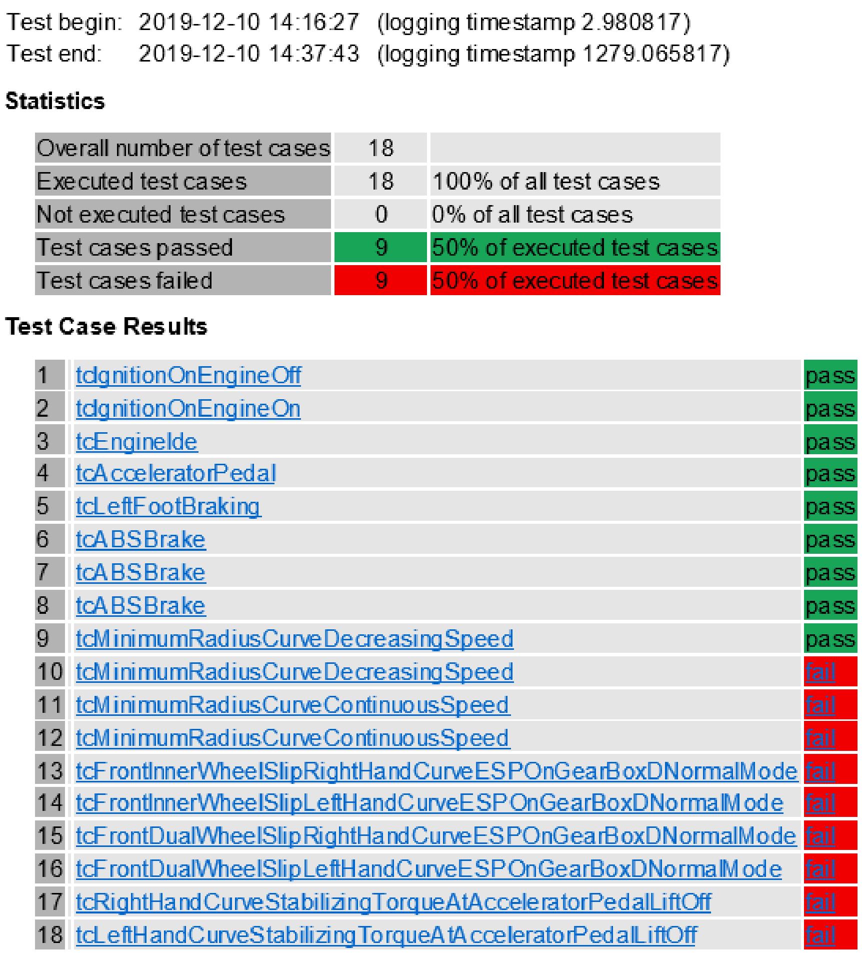
The third reason is the difference in evaluation method of test cases for wheel slip situation. In test cases 13, 14, 15, 16 the manual evaluation method was checking that the torque oscillated around a set point while the method used in automatic evaluation compared the wheel slip signal with the torque signal. The comparison between signals is stricter therefore only half of the evaluation intervals passed the automatic evaluation. It is also worth to know that the manual evaluation misses many evaluation intervals that are not easily seen.

5.2 Result

The result of this thesis work is a software which is able to evaluate test cases from a Safe Driving Test. This software can be integrated and used on different project at BorgWarner. The software takes log files of the Safe Driving Test as input and generates a test report with detailed information about the evaluation of each test case as output.

Depending on the size of the log file the evaluation time can vary. The evaluation time for running the test cases in the verification stage was approximately 20 minutes. Figure 5.11 shows the overview of the evaluation results for all run test cases. By clicking on each test case detailed information about the evaluation is shown. Figure 5.12 shows details in the evaluation of Test Case 4. The detailed test result consists of the result (pass or fail) from the evaluation of each relevant signal and a plot of the relevant signals. There is one signal called "iVerifying" which is set to one when it comes to the evaluation intervals and zero otherwise. This signal makes it easier for test evaluation engineer to see which parts of the log was evaluated.

The evaluation of Test Case 4 is carried out by comparing the similarity between the pump current signal and the accelerator pedal signal. Since the pump current signal has both delay and disturbances, the evaluation is performed using three algorithms: cross-correlation, moving average and correlation coefficient. The first step is finding the delay between pump current signal and accelerator pedal signal using cross-correlation. When the delay is found the pump current signal is shifted according to the delay so that it is time-aligned with the accelerator pedal. The second step is filtering the disturbances by running the moving average algorithm on the shifted pump current signal. This step makes the shape of the pump current signal stand out and makes the comparison more accurate. The last step is comparing the resulting signal from the previous step with the accelerator pedal signal using correlation coefficient algorithm.



Test begin: 2019-12-10 14:16:27 (logging timestamp 2.980817)
Test end: 2019-12-10 14:37:43 (logging timestamp 1279.065817)

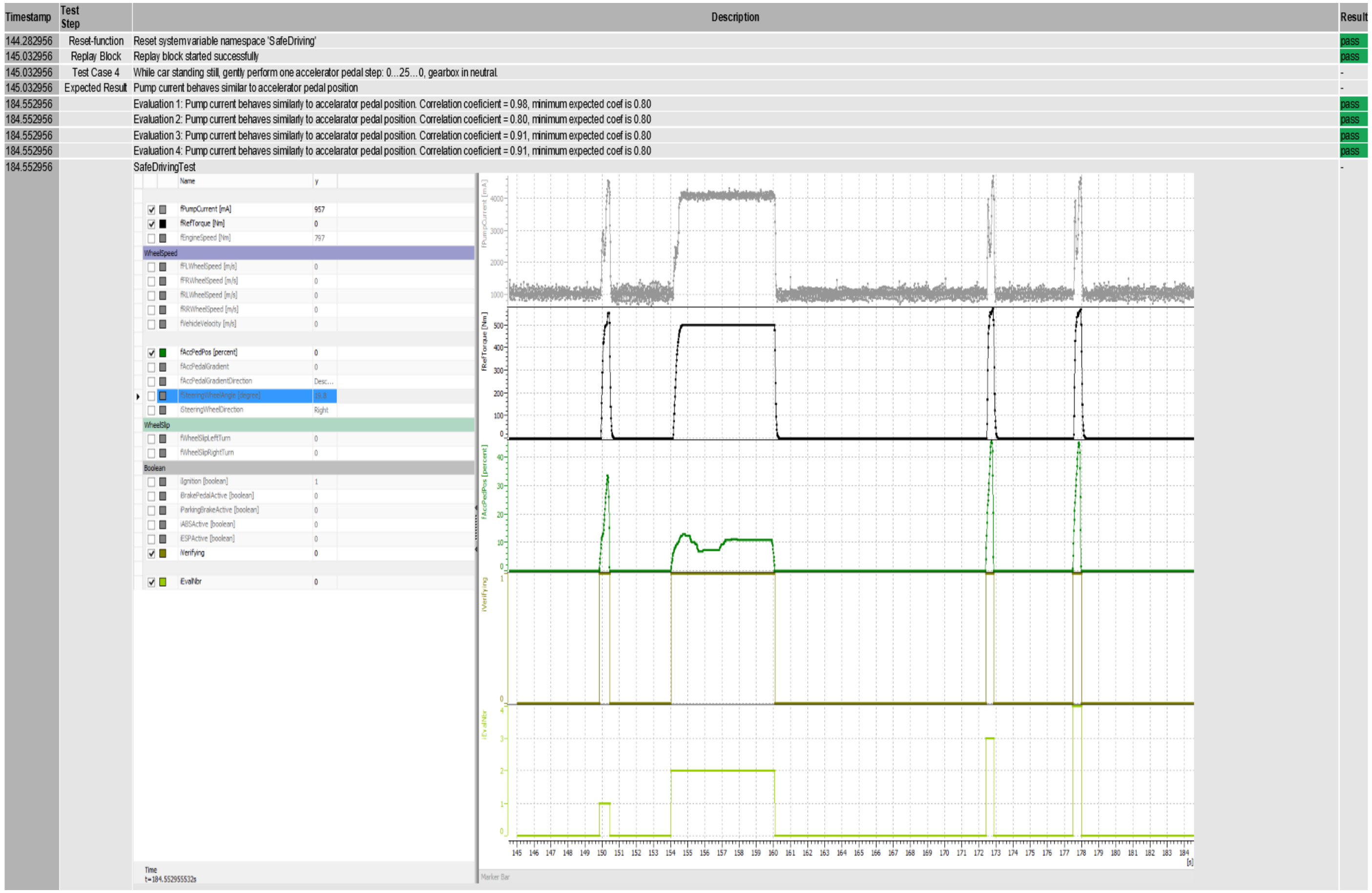
Statistics

Overall number of test cases	18	
Executed test cases	18	100% of all test cases
Not executed test cases	0	0% of all test cases
Test cases passed	9	50% of executed test cases
Test cases failed	9	50% of executed test cases

Test Case Results

1	tcIgnitionOnEngineOff	pass
2	tcIgnitionOnEngineOn	pass
3	tcEngineIde	pass
4	tcAcceleratorPedal	pass
5	tcLeftFootBraking	pass
6	tcABSBrake	pass
7	tcABSBrake	pass
8	tcABSBrake	pass
9	tcMinimumRadiusCurveDecreasingSpeed	pass
10	tcMinimumRadiusCurveDecreasingSpeed	fail
11	tcMinimumRadiusCurveContinuousSpeed	fail
12	tcMinimumRadiusCurveContinuousSpeed	fail
13	tcFrontInnerWheelSlipRightHandCurveESPOnGearBoxDNormalMode	fail
14	tcFrontInnerWheelSlipLeftHandCurveESPOnGearBoxDNormalMode	fail
15	tcFrontDualWheelSlipRightHandCurveESPOnGearBoxDNormalMode	fail
16	tcFrontDualWheelSlipLeftHandCurveESPOnGearBoxDNormalMode	fail
17	tcRightHandCurveStabilizingTorqueAtAcceleratorPedalLiftOff	fail
18	tcLeftHandCurveStabilizingTorqueAtAcceleratorPedalLiftOff	fail

Figure 5.11: The test report overview



Timestamp	Test Step	Description	Result
144.282956	Reset-function	Reset systemvariable namespace 'SafeDriving'	pass
145.032956	Replay Block	Replay block started successfully	pass
145.032956	Test Case 4	While car standing still, gently perform one accelerator pedal step: 0...25...0, gearbox in neutral.	-
145.032956	Expected Result	Pump current behaves similar to accelerator pedal position	-
184.552956		Evaluation 1: Pump current behaves similarly to accelarator pedal position. Correlation coeficient = 0.98, minimum expected coef is 0.80	pass
184.552956		Evaluation 2: Pump current behaves similarly to accelarator pedal position. Correlation coeficient = 0.80, minimum expected coef is 0.80	pass
184.552956		Evaluation 3: Pump current behaves similarly to accelarator pedal position. Correlation coeficient = 0.91, minimum expected coef is 0.80	pass
184.552956		Evaluation 4: Pump current behaves similarly to accelarator pedal position. Correlation coeficient = 0.91, minimum expected coef is 0.80	pass
184.552956		SafeDrivingTest	-

		Name	y
✓	▨	fPumpCurrent [mA]	957
✓	■	fRefTorque [Nm]	0
☐	▨	fEngineSpeed [Nm]	797
WheelSpeed			
☐	▨	fFLWheelSpeed [m/s]	0
☐	▨	fFRWheelSpeed [m/s]	0
☐	▨	fRLWheelSpeed [m/s]	0
☐	▨	fRRWheelSpeed [m/s]	0
☐	▨	fVehicleVelocity [m/s]	0
✓	🟩	fAccPedPos [percent]	0
☐	▨	fAccPedalGradient	0
☐	▨	fAccPedalGradientDirection	Desc...
☐	▨	fSteeringWheelAngle [degree]	19.8
☐	▨	iSteeringWheelDirection	Right
WheelSlip			
☐	▨	fWheelSlipLeftTurn	0
☐	▨	fWheelSlipRightTurn	0
Boolean			
☐	▨	iIgnition [boolean]	1
☐	▨	iBrakePedalActive [boolean]	0
☐	▨	iParkingBrakeActive [boolean]	0
☐	▨	iABSActive [boolean]	0
☐	▨	iESPActive [boolean]	0
✓	▨	iVerifying	0
✓	🟩	iEvalNbr	0

Time
t=184.552955532s

Figure 5.12: The detailed evaluation of Test Case 4

6 Conclusion and Future Work

The conclusion of this master thesis is that it is possible to automate the evaluation process of the Safe Driving Test, and that this automation will not only save time, but most importantly create a standard definition of how each test case should be evaluated.

Since it is the first version of the software and due to defects found in the verification stage, it is not advisable to totally trust the result provided by the software without a second verification of a test evaluation engineer. The evaluation should always be semi-automated at least on the very first versions. Semi-automated here means that a test evaluation engineer has to go through the test result to ensure that the evaluations are correct.

Comparisons between the automatic evaluation and the manually evaluation showed that the test result from some test cases was the same while other test cases failed in the automatic evaluation. Besides the failed test cases due to defects in software, it is worth to notice that the automatic evaluation is much stricter with higher accuracy compared to the manually evaluation, therefore it can discover unexpected results that can be missed in a manual evaluation.

The evaluation program was only a part of the new planned workflow that the software test department wants to achieve, thus there is a lot of work to be done.

As shown in Figure 1.2 the final goal is integrating the evaluation program to evaluate the Safe Driving Test performed by the simulated vehicle model. Currently, the evaluation program can only take the log file as input, so to be able to evaluate data from a simulation in real time the program must be further developed.

Since the program is only integrated into one project it is desirable in the future that the integration is done in more projects so that they also have access to automatic evaluation.

Due to the time frame of the master thesis I did not have time to fix the defects and do the desired improvements. To improve the reliability of the software it is necessary to fix all the defects and investigate further to improve the wheel slip detection. Currently the wheel slip detection can only detect wheel slip when the steering wheel angle is large enough. If the expected wheel speed for each wheel can be calculated, the wheel slip detection will be more accurate and manage to detect the wheel slip even when the steering wheel angle is small.

The solution of this project is implemented using custom algorithms. An alternative solution could be to use machine learning instead. The disadvantages of machine learning are that it requires a large amount of data with good quality and correct labelling for each test case. The requirements of the software might change over time, which means that the model needs to be retrained to take new requirements into account. However, new requirements do not have data that can be used as input to the training. A solution using logical rules on the other hand does not need large amount of data and is easier to adapt to changes of the requirements by modifying the logical rules.

Bibliography

Adenmark, M. (2003). *Automation of integration tests*. From Department for Signals, Sensors and Systems, Royal Institute of Technology. URL: `https://people.kth.se/~kallej/grad_students/adenmark_thesis03.pdf` (visited on 2019-10-31).

BorgWarner (2015). URL: `https://www.borgwarner.com/newsroom/press-releases/2015/04/20/borgwarner-wins-2015-automotive-news-pace-award-for-its-front-wheel-drive-electronic-limited-slip-differential-fxd-technology` (visited on 2020-04-24).

BorgWarner (2019). URL: `https://www.borgwarner.com/home` (visited on 2019-12-09).

Conrad, M., S. Sadeghipour, and H.-W. Wiesbrock (2005). "Automatic evaluation of ECU software tests". *SAE Technical Papers*. DOI: `10.4271/2005-01-1659`.

ElysiumAcademyPrivateLimited (2017). URL: `https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited` (visited on 2019-12-09).

Espfors, N. (2018). *CANoe - Simulink integration of vehicle model in existing test environment*. From Division of Industrial Electrical Engineering and Automation Faculty of Engineering,Lund University. URL: `http://www.iea.lth.se/publications/MS-Theses/Full%20document/5414_full_document.pdf` (visited on 2019-09-03).

Petersson, J. (2014). *Software test strategy*. Internal Document at BorgWaner. (Visited on 2019-12-13).

Steven W. Smith, P. (1999). *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, p. 277.

Vector (2019a). *CAPL scripting quickstart*. URL: `https://assets.vector.com/cms/content/products/VectorCAST/Events/TechNights/CAPLQuickstart_Generic_2018_Final.pdf` (visited on 2019-12-10).

Vector (2019b). *Logging formats*. URL: `https://kb.vector.com/entry/520/` (visited on 2019-12-10).

Vector (2019c). *Simulated environment*. URL: `https://www.vector.com/se/en-se/products/products-a-z/hardware/vt-system/#c8509` (visited on 2019-09-05).

Vector (2019d). *Testing ECUs and networks with CANoe*. URL: `https://www.vector.com/int/en/products/products-a-z/software/canoe/` (visited on 2019-12-10).

Vector (2019e). *vTESTstudio – Comfortable design of automated test sequences for embedded systems*. URL: `https://www.vector.com/int/en/products/products-a-z/software/vteststudio/` (visited on 2019-12-10).

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* April 2020 *Document Number* TFRT-6098
Author(s) Linh Nguyen Mai	*Supervisor* Måns Andersson, BorgWarner Mattias Wonizak, BorgWarner Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden Anton Cervin, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Automating the Evaluation Process of Software Testing in Vehicles

Abstract

The Safe Driving Test (SDT) is a software test performed in a real vehicle where the software is flashed to an ECU and connected to the CAN network in the vehicle. The test driver then conducts different driving scenarios according to the test instructions of the SDT. All the data during testing is logged and then evaluated manually by a test engineer. The entire process including testing, evaluating and fixing defects is very time-consuming and ineffective, especially the evaluation. Therefore the possibility of automating the evaluation process in Safe Driving Tests was explored in this master thesis.

The result of this work was a program containing an automatic evaluation process for the test cases in the Safe Driving Test. The program takes log files of the test cases as input and generates a report containing evaluations of each test case as output. This program can be integrated into different projects at BorgWarner.

The conclusion of this work is that it is definitely possible to automate the evaluation process and an automatic evaluation process will save time. However the test results must still be verified manually by a test engineer since the evaluation can only provide a pass or fail result and not the cause of the failed results.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-56	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/