Automated Robustness Simulation Testing of an Autonomous Vehicle

Master's thesis in Software Engineering

DAVID FOGELBERG & ELIAS HULT PAPPAS

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Automated Robustness Simulation Testing of an Autonomous Vehicle

DAVID FOGELBERG

ELIAS HULT PAPPAS

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Automated Robustness Simulation Testing of an Autonomous Vehicle
DAVID FOGELBERG & ELIAS HULT PAPPAS

Supervisor: Yue Kang, Computer Science and Engineering
Examiner: Eric Knauss, Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Automated Robustness Simulation Testing of an Autonomous Vehicle
David Fogelberg & Elias Hult Pappas
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Autonomous vehicles are facing a significant problem when it comes to testing different types of scenarios with various parameters, e.g. road friction and road slopes. It is crucial that autonomous vehicles can handle these scenarios to assure robustness of the system. In this study, this problem is addressed by developing a simulation environment for an autonomous vehicle model and testing the robustness of the model by applying one fault, steering miss alignment, and one condition, various road surfaces. Two algorithms, namely Search Based testing and Monte Carlo are involved in manipulating the values of these parameters, to find the best combination of parameters that gave the highest deviation values. The algorithms are later compared on how well they test the robustness of the autonomous vehicle model by comparing these deviation values.

Keywords: Testing, Automation, Robustness, Autonomous vehicles, Autonomous, Simulation, Monte Carlo Algorithm, Genetic Algorithm, .

Acknowledgements

We want to thank our supervisor Yue Kang and Christian Berger for guidance and the opportunity for being able to conduct this study.

David Fogelberg & Elias Hult Pappas, Gothenburg, October 2018

Contents

1
Introduction

The development and production of autonomous driving are rapidly growing which is creating new challenges for testing and validation. Testing and validating autonomous vehicles are essential to guarantee safety for drivers and pedestrians. However, this is difficult when it comes to testing and validating autonomous vehicles for real scenarios. It is often challenging to locate different road environments with various characteristics and to reproduce multiple scenarios, which is time-consuming and costly (Betts and Petty, 2016). Therefore this study is conducted in order to integrate a simulation environment for autonomous vehicles.

The simulation environment is used, for testing the robustness of an autonomous vehicle model by simulating a scenario, which consisted of various conditions. The autonomous vehicle model needs to pass this scenario and then validated before being tested in reality. The simulation environment also saves time and cost for testing and validating real autonomous vehicles since scenarios can be simulated and manipulated beforehand. In the context of the thesis, an analysis is done with two test generation methods by comparing their performance of how well they can test the robustness of the autonomous vehicle model, by manipulating one fault and one condition during the simulation runs (Betts and Petty, 2016).

1.1 Statement of Problem

Autonomous vehicles are facing major challenges when it comes to testing various types of scenarios with different conditions e.g. road friction, road slopes, obstacle avoidance, and regulation of various parameters (Baekgyu, et al, 2016). It is essential for an autonomous vehicle to manage all these scenarios, in order to guarantee a robust system (Berger, 2014).

Today, testing and validating all of these scenarios are done by regulating parameters and applying it to physical models. Testing and validating the outcome in a massive scale is very complex, costly and time-consuming, since scenarios e.g. require various traffic flow and specific obstacles (Baekgyu, et al, 2016).

By integrating a simulation environment using a virtual environment, testing and validating all of these scenarios can be done virtually on a massive scale. Furthermore, testing time and resources can also be reduced, for validating various scenarios compared to field experiments on a real vehicle since a simulation does not need to be exposed to real-world scenarios (Masuda, 2017). In the simulation environment, a specific scenario can be launched and reproduced for several amounts of runs, which will potentially increase the robustness of an autonomous vehicle system. The scale of each simulation run is limited by 47 seconds for each inserted test case in the simulation environment, whereas a test case contains path coordinates, one fault, i.e., steering miss alignment, and one condition, i.e., road surface for the specified scenario.

In addition, testing scenarios manually in a virtual environment is time-consuming and expensive (Bogdan, 1990). In the real world, an autonomous vehicle requires large inputs of test cases that contains multiple scenarios with different conditions, which is necessary to guarantee a robust system (Betts and Petty, 2016). Therefore in this study, two test generation methods, Search-based testing (SBT) and Monte Carlo testing (MCT), are used to solve the problem of applying test cases in a scenario for testing the robustness of an autonomous vehicle model. This is done by using a simulation environment and inserting test cases containing the fault, steering miss alignment, and the condition, road surface, and by manipulating the values of the fault and the condition, as well as their coordinates. The algorithms manipulate the fault, steering miss alignment, by using values ranging from -0.05 to 0.05, and the condition, road surface, by reducing the friction using the values ranging from 1.0 to 0.1.

1.2 Purpose of Study

The purpose of the study is to develop a simulation environment for the Revere Lab at Chalmers University and to test the robustness of an autonomous vehicle model by exposing it to a scenario with two parameters (reduced friction and steer-

ing miss alignment) in the simulation environment. The simulation environment is integrated by using the cross-platform VDrift and the middleware OpenDaVINCI. The autonomous vehicle model's robustness is measured by calculating the model's deviation after inserting the manipulated test cases which contained the condition road surface and the fault, steering miss alignment.

In addition to that, manually application and manipulation of the test cases would be an immense labor-intensive task. Automated Test Generation (ATG) is a process to deal with this and it automatically generates tests and executes the software in a particular path (Bogdan, 1990). The test case generated needs to be an optimized selection of the all existing combinations to best test the robustness of the autonomous vehicle model in the scenario. There are several algorithms to accomplish this; in this study, one SBT algorithm which portraits natural selection and one randomized algorithm MCT (Betts and Petty, 2016) are used for applying and manipulating the test cases.

In order to achieve appropriate study results regarding the robustness of the autonomous vehicle model, the two test generation methods are compared on how well they test the robustness of the autonomous vehicle model with respect to deviation values. According to Betts and Petty (2016), SBT should outperform MCT by generating higher deviation values when the simulation runs increase, whereas larger deviation values indicate poorer robustness. The purpose of this study also includes validation of such theories by comparing it with the research results.

Previous work regarding verifying and validating the robustness of an UAV model (Betts and Petty, 2016) and an UAV model controller (Schultz, Grefenstette, and De Long, 1993) has been done, by using the algorithms stated above. Both of the previous studies showed that SBT is the most efficient test generation method for robustness testing the stated models. On top of that, this thesis focuses more on performing the testing process on the model in a simulation environment with scenario visualization, which has not been implemented in the previous approaches. The scenario visualization is a requirement from the Revere Lab since their current simulation environments did not have this functionality.

1.3 Limitations and Delimitations

The limitations and delimitations for testing the robustness of the autonomous vehicle model in the simulation environment are brought up in this section. Due to the time limitation and the time of running one test run, only one scenario was used for testing the robustness of the autonomous vehicle model. The number of test runs was limited up to 80 runs and only two test generation methods were used, Monte Carlo testing (MCT) and Search-based testing (SBT).

The vehicle model itself was developed by a third party and no model was developed in this study. This is for two reasons, one for compatibility reason and the other

one for testing robustness off an unknown vehicle model.

2
Background

This chapter will provide a literature framework showing the importance and necessity of this study. The literature has been selected and analyzed to fit the research purpose and serve as a baseline for the research questions. It is essential for the reader to understand this chapter in order to apprehend the research method and study results.

First of all, it explains the value of having an robust system that can handle all traffic situations during its lifetime. The next section discusses testing autonomous vehicles with simulations e.g. testing process etc. These two mentioned sections are crucial for the reader to understand since they explain the reasons behind the conduction of this study. In addition, it is essential for the reader to understand how test generation algorithms operate. Therefore in the automated testing section, Monte Carlos and Search-based testing are described in more detail. Furthermore, the chapter defines the difference between situations, scenes, and scenarios and different testing abstraction levels, which guides the reader in comprehending the testing level this study focuses on. Lastly, the selected platforms used for creating the simulation are presented and a motivation for selecting them.

2.1 Robustness Testing of Autonomous Vehicles

The importance of a robust system in an autonomous vehicle is necessary to guarantee traffic safety for passengers and pedestrians (Berger, 2014). To ensure this, robustness testing is essential, which is defined as *how well the software performs under stressful environmental conditions or how well the software handles invalid insertions* (Lei, et al, 2010). Robustness testing of autonomous vehicle systems are highly dependent on data from its surrounding, which means testing is time-and-resource consuming due to lack of proving grounds and testing equipment (Berger, et al, 2013). In addition, autonomous vehicle systems are becoming more complex and hence it is a challenge to validate these systems in real road tests since it would require millions of kilometers with different testing grounds. Without these testing grounds, it would not be possible to validate and obtain suitable statistical system performance measurements (Zofka, et al, 2016). Therefore, robustness simulation testing is crucial for validation of autonomous vehicle systems and to reduce testing time, resource costs, etc.

In order to replace a driver and ensure safety, the autonomous vehicle system must be able to handle all traffic situations which can be different depending on country location. These traffic situations, often called scenarios, can be complex to validate for an autonomous software system due to possible demand for space and repeatability, e.g. specific highways. In this case simulating scenarios with different parameters is necessary in order to test and validate autonomous system algorithms for robustness and quality, where simulations rely on realistic vehicle models (Zofka, et al, 2016; Berger, 2014). Still, the challenge of implementing simulation results to a real vehicle can be difficult because of syntactical or semantic incompatibilities on software/software or software/hardware level, which can delay the experiments (Berger, 2014).

Simulation in virtual environments has been proven successful in conceptualizing and evaluating algorithms for autonomous vehicles. In order to achieve a successful simulation environment, it must include models for producing images from the virtual surroundings of the vehicle, models for producing data for distance-based sensors and physical motion model for the vehicle dynamics to enable a closed-loop experimentation system (Berger, 2014).

2.2 Testing Autonomous Vehicles with Simulations

Testing autonomous vehicles through simulation is essential for reducing cost, time and increasing robustness (Masuda, 2017). There exist however critical issues regarding simulation testing. Testing inputs can be very large since they could contain scenarios with other vehicles, traffic lanes, and pedestrians, etc. This makes it very difficult to cover all of these scenario cases when testing an autonomous vehicle in a simulation. The amount of time it would take to cover all these cases is time inefficient and costly (Masuda, 2017), which is known as exhaustive in the soft-

ware engineering industry (Kuhn, Wallace and Gallo, 2004). According to empirical research regarding quality and reliability, software errors are caused by a subset of test cases. If with certainty these errors were triggered then testing more test cases with discrete and continuous values would be exhaustive. In reality, this is extremely complex to achieve and therefore software testing should be based according to empirical data on errors from software systems with resembling domains (Kuhn, Wallace and Gallo, 2004).

There exist however different kinds of testing methods which can be used to solve these issues partially (Masuda, 2017). Typical examples are Random testing whereas Monte Carlos and Search Based testing are included, Combinatorial testing techniques and Scenario testing. Random testing can be applied to the test design by using random items and values in the test cases for the simulation. However, since the number of bugs in the software is unknown, it is difficult to guarantee full coverage testing and therefore random testing should only be used partially.

The testing process should be as follows:

1. Set road information according to a scenario for example a highway with no-lane change or traffic lights.
2. Define values in the scenario for example other cars', pedestrians and prepare their information (Zofka, et al, 2016).
3. Choose a test design technique for example random testing (Masuda, 2017).
4. Create test cases, for example random values. It is important here to prioritize test cases which can cause most harm and vulnerability to passengers, pedestrians, the vehicle itself etc.
5. Execute the test cases in the simulation environment.
6. Obtain the test results from the simulation for example vehicle position, speed, acceleration, deceleration etc.
7. Evaluate the testing results according to for example vulnerabilities, safety, etc.

2.3 Automated Testing

Software testing is a costly task and often accounts for half of the development cost. If it were to be automated, the cost of developing would have a significant reduction (Hooda and Chhillar, 2014). Software test automation is a process for accomplishing this by automating so-called testing activities, where the activities include, e.g., creating the testing database or Oracle, test case generation, test case execution, result comparison and verification, evaluation and assessment, and reporting results (Alsmadi, 2013). Figure 2.1 displays a framework of the mentioned testing activities, followed by a description of feasible automation for each activity.

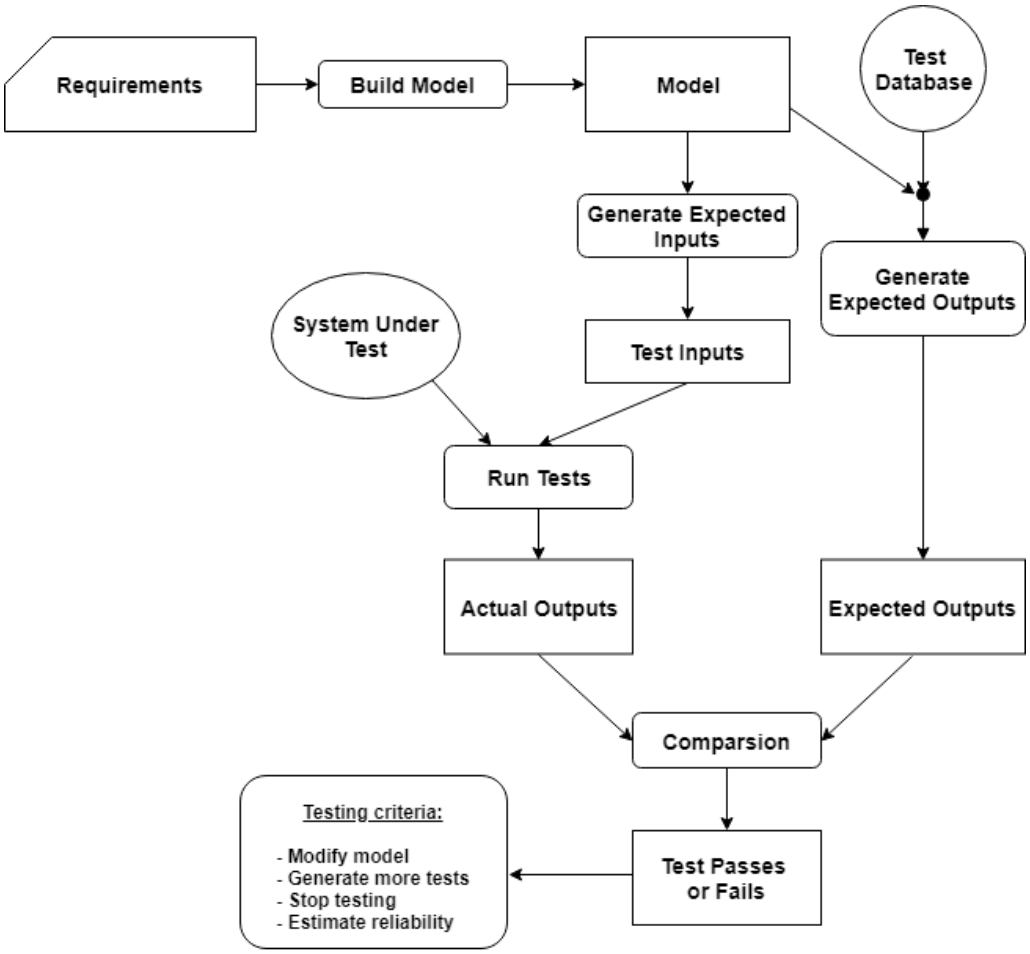

Figure 2.1: Software automation testing activities (Alsmadi, 2013).

1. Building a test model includes the software requirements specification. The specification documentation can be informal or formal (Alsmadi, 2013). Although, automated test case generation sometimes requests that the requirements are formally specified since they need to be structured and persistent, to be able to create or use an automated test case generating tool. Another option without the need for formal requirements is to generate test cases automatically from the application itself.

2. The testing database contains all of the generated test cases and their predicted results, which are, e.g., used in regression testing when an application has been updated to confirm that the altered code has not produced any errors in the existing code.

3. Test case generation includes test case inputs and expected outputs, which can be automated by using formal requirement specifications or using the application itself. In the application various test generation method can be implemented, e.g., fuzzy logic, finite state machine or genetic algorithms (Hooda and Chhillar, 2014).

4. Test case execution is the most popular activity for automating since there exist commercial tools, e.g., JUnit and NUnit, which makes it easy to produce test cases on the code without the need of a user interface (Alsmadi, 2013).

5. Test case results comparison and verification is one of the most complex to automate due to the complexity of describing expected test case results. An example of this is managing the code so that it automatically identifies a GUI state after launching a test case which changes the color of the GUI components.

The context of this thesis mainly focuses on automating test case generation accomplished by using testing generation methods. The next section introduces a more detailed description of test generation methods.

Test Generation methods

Automated test generation (ATG) methods automatically identify and generate test cases and test data that satisfies the testing criteria. A tool or an algorithm is required to assist in generating test cases and test data for a software (Hooda and Chhillar, 2014; Bogdan, 1990). Figure 2.2 displays a general process of using a test generation method. The test generation method generates test cases to the software application under test that launches the test cases. After executing the test cases, the application sends their output to the test generation method. Depending on the usage of test generation method, test cases can be selected to a subset, minimized to a subset, prioritized, and evaluated, as well generate new test cases (Hooda and Chhillar, 2014).

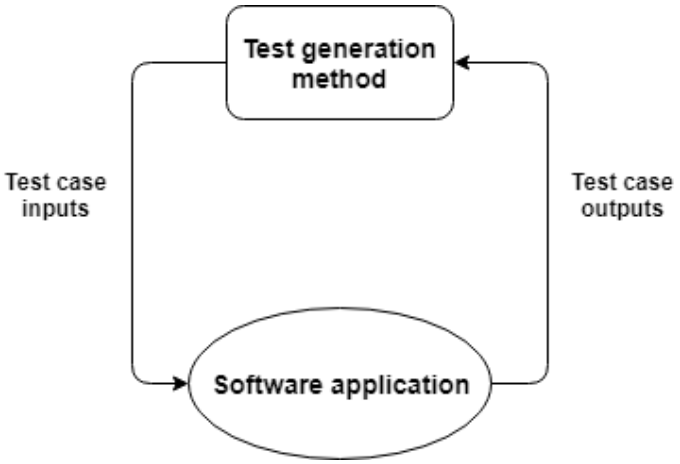

Figure 2.2: Test generation process.

There exists various test generation methods, whereas three testing generation algorithms will be discussed. To thoroughly test a software, full combinatorial testing is used, which produces an entire population as samples of a test data set. However, due to the huge size of test cases for autonomous vehicles, it is not feasible to test

all of them in a reasonable amount of time (Betts and Petty, 2016). Different algorithms could be applied in order to decrease the amount of test case samples, among which Monte Carlo is commonly involved in such cases by randomly generating a sample of test cases from a population. Another algorithm is the Genetic algorithm Search-Based Testing. It is frequently used on optimization problems with multiple input variables and has been used in abundant cases for model testing. The next two sections explain in detail how the two automated algorithms in this study operate.

Monte Carlo testing

The Monte Carlo testing method operates by running random samples consisting of pseudo-random numbers from a population of values (Thomopoulos, 2013). In this study, the sample consists of test cases and random fault-and condition values, as well random coordinates. During each run, random variates of input variables are generated which come from specified probability distributions. *A random variate is defined as a random outcome from a random variable*, which can be either continuous or discrete. Continuous random variates can adopt any value in a specific interval and discrete random variates can adopt a specific list of values. Since the input is always random when using Monte Carlo testing, the output is always random.

One popular distribution formula is called uniform distribution, which can be used for discrete or continuous variates (Thomopoulos, 2013). A continuous uniform distribution is when a variable x with parameters (a, b) has an equal probability of falling anywhere from a to b (see uniform distribution formulas). Furthermore, a discrete uniform distribution is when a variable x becomes all integers from a parameter interval, e.g. a to b, with equal probabilities. In addition, normal distribution is a distribution commonly used for applications. It is symmetrical and has a bell-shaped density, where mean is μ and the standard deviation is σ. For a standard normal distribution, μ is usually zero and σ is 1. However, in this study, Monte Carlo is used with a continuous uniform distribution and therefore it is necessary for the reader to comprehend the formulas listed below.

The uniform distribution formulas listed below are essential in this study for comprehending how Monte Carlo operates in the simulation environment.

Continuous uniform:

The probability density of x is:

$f(x) = 1/(b-a)$ for a <= x <= b

Cumulative distribution function:

$F(x) = (x-a)/(x-b)$ for a <= x <= b

Expected value is calculated by:

$E(x) = (b + a)/2$

Variance of x is calculated by:

$V(x) = (b - a)^2/12$

How to spawn a random continuous uniform variate of x:

1. Spawn a random uniform $u \sim U(0, 1)$

2. $x = a + u(b - a)$

3. Return x (Thomopoulos, 2013).

Example

The following example shows how to generate a random variate of x.
Assume that x is a continuous uniform with an interval (5, 15).

1. Produce a random uniform $u \sim U(0, 1)$, e.g. u = 0.54

2. $x = 5 + 0.54(15 - 5)$

3. $x = 10.4$

Discrete uniform:

The probability of x becomes:

$p(x) = 1/(b - a + 1)$ for x = a to b

The cumulative distribution function:

$F(x) = (x - a + 1)/(b - a + 1)$ for x = a to b

Expected value is calculated by:

$E(x) = (a + b)/2$

Variance of x is calculated by:

$V(x) = [(b - a + 1)^2 - 1]/12$

How to spawn a random discrete uniform variate of x:

1. Spawn a random continuous uniform $u \sim U(0, 1)$
2. x = ceiling$[(a - 1) + u(b - a + 1)]$, ceiling = round of to closest integer.
3. Return x (Thomopoulos, 2013).

Example

The following example shows how to generate a random discrete uniform variate of x. Assume that x is a discrete uniform which includes the parameters (15, 25).

1. Spawn a random continuous uniform $u \sim U(0, 1)$, e.g. u = 0.45

2. x = ceiling $[(15 - 1) + 0.45(25 - 15 + 1)]$

3. $x = 19$

Search Based testing using GA

Search based testing uses a fitness function in order to search a test objects input domain, which automatically finds test data. A search based testing method can be based on different algorithms, e.g. Genetic algorithms (Harman, et al, 2010). The algorithm is described in more detail below:

Genetic algorithms are part of the evolutionary family, which uses mechanisms inspired by the processes of Darwinian evolution. It performs a global search on a generated population of individuals to find the best solution candidate. The implementation of a Genetic algorithm requires several stages (see Figure 2.3), which are dependent on the outcome of test data (Harman, et al, 2010). The stages are listed and described below:

1. A random population of solution candidates known as individuals, e.g. test cases, are initiated (Betts and Petty, 2016; Harman, et al, 2010), which are encoded as bit strings. After the generation of individuals, the rest of the stages are executed in a loop.

2. The genetic algorithm evaluates individuals with a fitness function, in order to find a requested solution, i.e., the largest deviation value. An alternative is to use a fitness function of assessing various criteria (Harman, et al, 2010). In this study, the fitness function also evaluates the criterion of allowed simulation runs.

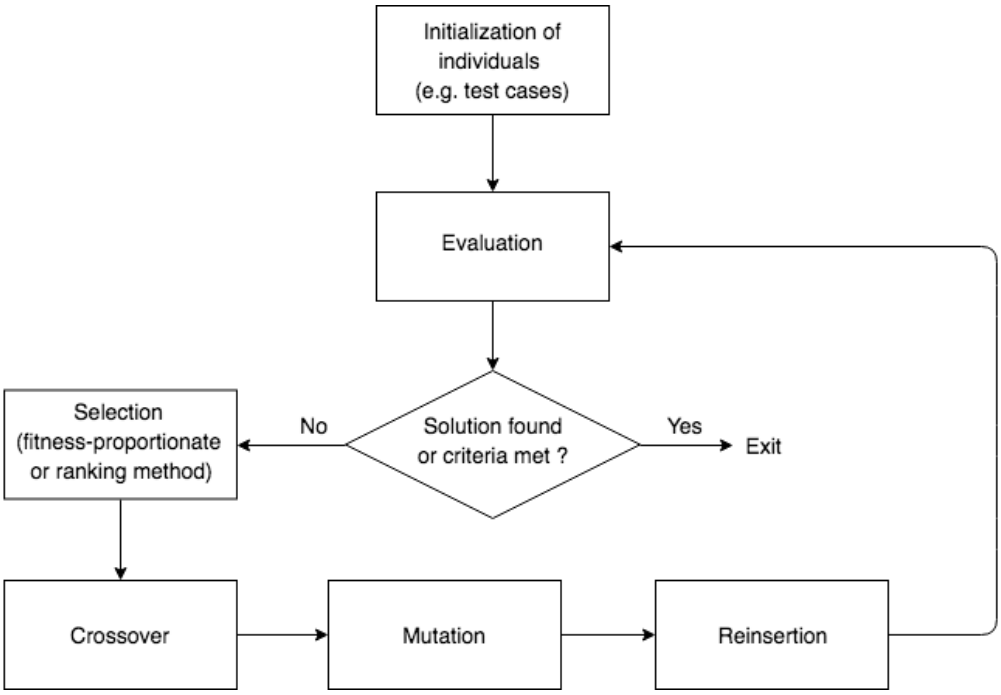

Figure 2.3: The stages of a Genetic algorithm (Harman, et al, 2010).

3. Individuals are selected as parents for the crossover stage. The selection can be done in several ways, for example by a probability (P_c) that is proportionate to an individual's fitness value, which is called fitness-proportionate selection. Furthermore, individuals can be selected according to ranking selection methods. Individuals are ranked and ordered according to their fitness value (Harman, et al, 2010). However, in this study individuals, i.e., test cases, are ranked by deviation values instead of fitness values. They are then selected randomly at a probability corresponding to their rank.

4. Parent individuals are spliced by a crossover-operator, e.g. simple one-point crossover, to create offspring solutions. In a simple one-point crossover, two parents genes, e.g. bit strings, are spliced in order to create two children (see Figure 2.4). The two children will have a mix of the two parent individuals, i.e. ones and zeros, and will move on to the mutation stage (Harman, et al, 2010).

Figure 2.4: Two bit strings are spliced at position 3 by an single-point crossover (Harman, M. et al, 2010).

5. Offspring individuals are randomly modified to create variety into the search, which is known as mutation. Each individual's chromosome, i.e. a string, bits are

randomly flipped according to a certain probability (P_m). The probability is usually measured by 1/len, whereas len is the length of the string (Harman, et al, 2010).

6. A new population is created from the generated offspring and the current population. One approach is to include only the generated offspring individuals, while another is to include the best individuals, where the weakest offspring individuals are replaced. The newly created population is then sent for evaluation and the loop repeats (Harman, et al, 2010).

2.4 Testing Abstraction Levels for Vehicles

In this section, four kinds of testing procedures on different abstraction levels are introduced and described in more detail. These include Unit testing, Situation-based testing, Scenario-based testing and Real-world driving testing. The testing procedures are essential when it comes to testing autonomous vehicles on different abstraction levels. It is important to understand how these testing procedures are connected since this research main focus is on scenario-based testing (Ulbrich, et al, 2016).

Unit Testing

Unit testing is a popular method for testing various software in software engineering, which consists of different steps i.e. preparation, obtain test suite, launch and analyze tests (Gelperin, et al, 1993). It is a basic level of testing which reduces errors in code functions. The functions could be calculations of distance between vehicles, time gaps, time to collisions etc. However, the testing method is not suitable when it comes to validating situation functions. These functions often require knowledge from the past development of a situation, which means unit testing does not meet the criteria for this type of evaluation. Here is where situation based testing comes into play (Ulbrich, et al, 2016).

Situation Testing

Situation-based open-loop testing is much wider then Unit testing since it doesn't focus on testing single functions and lines of code (Ulbrich, et al, 2016). In situation based testing, a data structure is set up as a mock-up for a real driving situation. This mock-up is included in a test-suite which consists of test cases for the situation, situation parameters, and pass-fail criteria. The test-suite is used to test a tactical behavior planning module and can be updated with new test situations. The tactical behavior planning module needs to handle specific test situations and is evaluated based on pass-fail criteria. Depending on the planning module's tactical driving decision, it either gets a pass or a fail. The test situations can be repeatedly simulated which increases the robustness of the planning module. This means that states of dynamic and model-free filtering components can be reached and tested.

There are some limitations with situation based testing. It it insufficient for testing components containing model-based filters, since the situation prediction model is unable to predict the current situation under temporal development. Furthermore, it is limited by a situation not being modified, a module under test might not behave correctly if a situation is modified and make an incorrect decision (Ulbrich, et al, 2016).

Scenario Testing

Testing autonomous functions are essential for preventing malfunctions in software since it can cause harm to the vehicle, driver or pedestrians. The tests could be conducted in real road environments however this is inefficient for several reasons. First of all, exposing a vehicle for test scenarios does not scale well since it requires different road environments that have various characteristics. Locating these road environments can be difficult and requires a lot of time (Baekgyu, et al, 2016).

Furthermore, reproducing certain scenarios can be difficult since it requires the same road environment with the same characteristic. Therefore, it is difficult to expose modified software to the same scenario that caused a malfunction and trace it back for validation. Lastly, software under progress development is not guaranteed safe and can cause harm to the vehicle, the driver and pedestrians (Baekgyu, et al, 2016). In order to solve this issue, scenario-based testing should be performed and validated.

Scenario-based closed-loop testing has less limitations than situation based testing. The test-suite includes test cases which contains entire scenarios, scenario parameters, and pass-fail criteria. The scenario-based testing covers scenes since a scenario is a sequence of scenes, well as events to modify the scenes. Goals and values are used to derive situations and for driving input (Ulbrich, et al, 2016).

In addition, in this case, there is not only one module in the system under test but several, which is a significant difference compared to situation testing. Several models from the autonomous vehicle are tested as a whole by modifying scenes. The scenes are modified according to the behavior and control from the vehicle's driving functions. The driving functions are influenced by prediction models in the system, which makes the vehicle behave a certain way when being exposed to a scenario. Furthermore, the modification of scenes during simulation makes it possible for objects to move ahead initiate maneuvers over the driving course. The maneuvers are controlled by driving models in a vehicle or by events from a scenario. This is not possible in situation based testing since the situations are not modified during the simulation (Ulbrich, et al, 2016).

The greatness of scenario-based testing makes it possible to simulate complex scenarios, which are almost impossible to test and simulate in reality. This makes it possible to test interaction of several modules at once and to find signal latencies and find functional instabilities.

There are some limitations with scenario based testing. In order to have accurate testing, the models needs to behave as the models in a real vehicle. However, these models in the real world are complex and this makes it hard to create models in a simulation which behaves identically to real models. Often models in a simulation are tailored in order to work in the simulation environment, but would not be accurate enough for the real world. Furthermore, it is time-consuming to create accurate models which slow down the development (Ulbrich, et al, 2016).

Real World Driving Testing

Real world driving testing is the last step of the testing procedures. This type of testing should only be conducted if all tests from scenario-based testing and situation based testing have passed. Here the simulation model is verified and exposed to acceptance testing. It is only in the real world where a vehicle's driving functions can be tested against real conditions and uncertainties. Therefore this is an important last step for autonomous vehicle testing (Ulbrich, et al, 2016).

When testing real autonomous vehicles, it is required to have test-cases with scenarios, scenario parameters, and pass-fail criteria. In this case, the parameters are indirect since you can't choose them from the beginning. Driven trajectories and traffic participants will be arranged in a certain way, depending on the current situation. Therefore the test will be random when testing in the real world. The benefits of real-world driving test is that it does not require any simulation model and therefore no errors are introduced from incorrect models. However, there are some limitations with real driving testing. The test-cases are random, which mean they are not reproducible since the behavior of traffic participants are random. Furthermore, the test-cases are costly and time-consuming since they cannot be executed faster than real-time (Ulbrich, et al, 2016).

2.5 Scene, Situation and Scenario

In order to know the difference between different test levels when testing autonomous vehicles, it is important to know the difference between a scene, a situation, and a scenario.

A scene includes moving elements or elements that are capable of moving (dynamic elements), actors, observers self-representations and relationships among those entities. These are all included in a snapshot of an environment which is described by a scene. A scene is uncertain, incorrect and incomplete in the real world (Ulbrich, et al, 2016). According to Geyer, et al, (2013), the dynamic elements mentioned above have predefined conditions that consist of behavior, starting location and speed. Although the vehicle itself and the driver have a predefined location, they are not included in a scene since they do not have predefined behavior. Figure 2.5 shows an example of a scene which consists of traffic lights and a preceding vehicle.

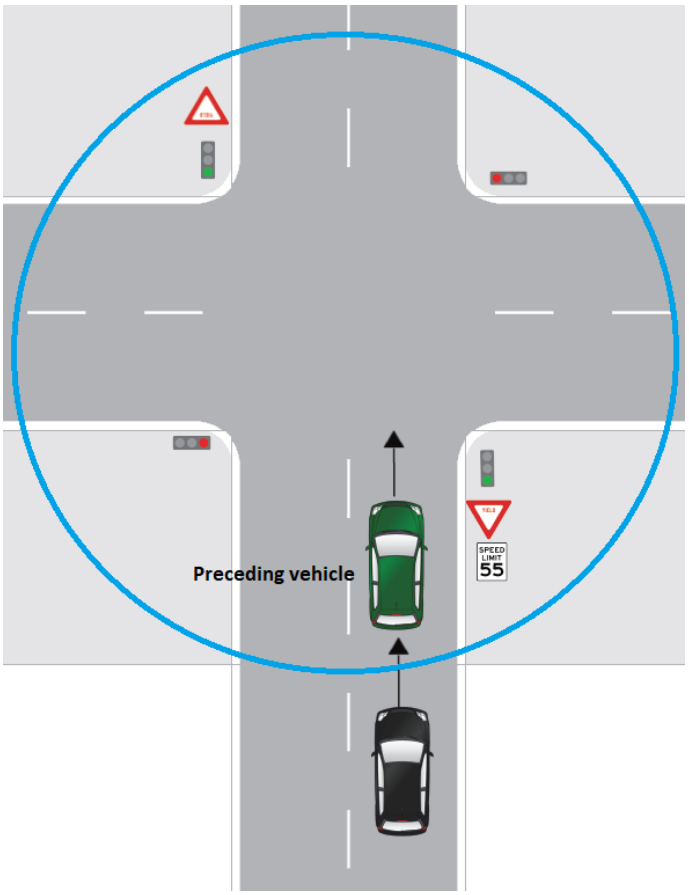

Figure 2.5: Scene with traffic lights (Geyer, et al, 2013).

A situation is an extraction from a scene and is an entirety of circumstances, whereas an appropriate behavior of a vehicle needs to be considered at a particular point of time. A right behavior pattern needs to be established by interpreting a selection of information, goals, and values derived from a scene. It could be for example a scene whereas there is a paper bag flying towards the vehicle and a playing child. The child needs to be considered here as the highest value and therefore the vehicle should behave accordingly to fulfill the goal of not harming the child (Ulbrich, et al, 2016).

A scenario includes situations and a series of several scenes, where there is a starting scene and an end scene. Goals, values, actions, and events can be specified accordingly in a scenario (Ulbrich, et al, 2016). Figure 2.5 displays an example of a scenario where the vehicle plans to turn right. In this case, various events can occur, e.g. the vehicle might accelerate, then decelerate and then turn right, which generates three situations (Geyer, et al, 2013).

Furthermore, a scenario spans a certain amount of time which is not the case in a scene, as mentioned by Geyer, et al, (2013) the vehicle itself and the driver is not part of the scene, which makes it difficult to know when the scene ends. In addition, when testing autonomous vehicles with software testing, scenarios need to

be established and generate specific test cases. These test cases need to have specific parameters and needs to be mapped to a specific scenario (Ulbrich, et al, 2016).

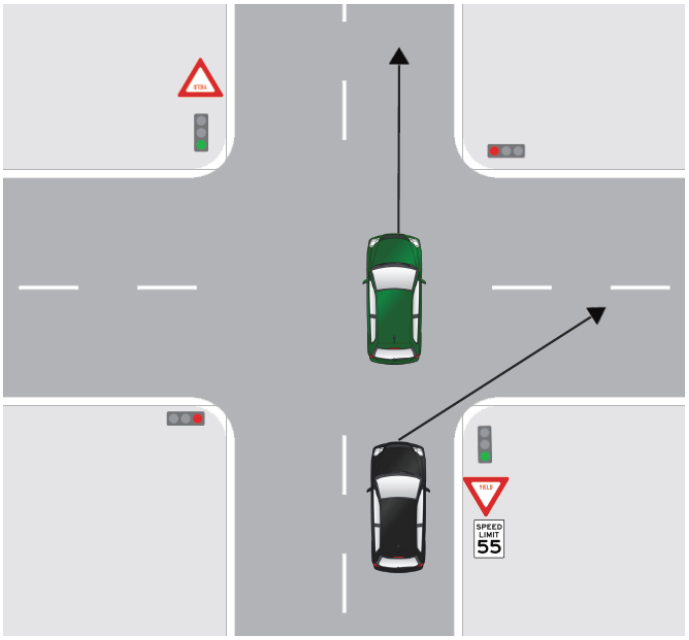

Figure 2.6: Crossway scenario (Geyer, et al, 2013).

The mentioned information above gives the reader an insight into how to distinguish between scenes, situations, and scenarios. By using the information, specific test cases can be developed from particular uses cases with scenes, scenarios, and parameters. This enables testing and achieving specific behavior patterns for autonomous vehicles, which they need to adopt when being exposed to various scenarios (Ulbrich, et al, 2016).

2.6 Platforms

In this section three platforms that the study required for developing the simulation and executing tests are presented. The first one is Docker which is a container platform that makes it possible to run software projects in different operating systems. Furthermore, the second one is Blender which made it possible to create 3D models for the simulation. Lastly, VDrift is presented which is used as the core simulation environment for this study.

Docker

Docker is a container platform which makes it easier when developing, packaging and running portable applications (Vohra, 2016). It is often a challenge to include dependencies for different kinds of software applications, which could be for example runtime libraries and system tools. When using Docker, the developer doesn't need

to worry about this since Docker handles this automatically. This is done by including all the dependencies in a so-called Docker image which can be executed on any platform and environment. A Docker image runs within the OS kernel and provides a container which the Docker software will be run in. The Docker container is an isolated environment which includes all that is required for running a software on different OS. The container has its own filesystem and environment variables which makes it independent on OS.

The code snip below shows an example of how a Dockerfile can look like. In this example, Python 2.7-slim is the version used for running the software. Furthermore, the working directory is set to /app, which is where all the source code is located. The directory content is then copied to the container by using the ADD command. If the required packages are not available on the current OS, the RUN pip install command will download the required packages. In order to make it available for everyone, set the Port to 80 by using the EXPOSE command and also define an environment variable by using the ENV command. Use CMD ["language", "filename"] to run the software when the container is launched. Se example below for further explanation (Docker, 2018):

```
FROM python:2.7-slim
WORKDIR /app
ADD . /app
RUN pip install –trusted-host pypi.python.org -r requirements.txt
EXPOSE 80
ENV NAME World
CMD ["python", "app.py"]
```

Blender

Blender is a 3D open-source modeling software which supports the creation of models, animation, rigging, simulation, rendering, compositing and motion tracking, video editing and game creation. Furthermore, it supports Python scripts which can be used to create more advanced applications and tools (Blender, 2018).

VDrift

VDrift is a open-source driving simulation cross-platform which was made with car drifting in mind. The platform will serve as the core virtual environment and will be used to create the simulation environment. The reason for choosing VDrift: It is Open Source under GPL v2 license, it is a light-weighted environment for simulation, it has multi-platform support, easy access to data and customized vehicle models and the compatibility of cooperating with the current middleware OpenDavinci in the lab, which was a requirement from the Revere Lab. See Table 2.1 for comparison with other virtual environments (Kang, et al, 2018).

Name of the sim. env.	Accessibility	Typical use-cases	Compared with VDrift (our choice)
VDrift	Open source (GPL v2)	3D car racing simulation Driving physics ML related to self driving	
TORCS	Open source (GPL v2)	3D car racing simulation Programmable AI for racing	Lack of simulation details (Precise friction, etc.)
Gazebo for ROS	Open source (Apache 2.0)	Robot dynamics simulation 3D visual environment Sensor data generation	Focusing more on robotics other than vehicle dynamics
MS AirSim	Open source (MIT License)	Drone and car simulation 3D visual environment HIL controller support	Focusing more on drone (Possible alternative in the future after several updates)
CARLA	Open source (MIT License)	3D urban environment Camera and sensor simulation	Lack of multi-platform support
SCANeR Studio	Commercial software	Traffic scenario simulation Vehicle dynamics Autonomous driving	Not open source, expensive to be afforded

Table 2.1: Lists the difference between virtual environments and show why VDrift was chosen (Kang, et al, 2018).

3
Related Work

In this chapter, the result of four previous case studies are presented, three regarding Monte Carlo and Search-based Genetic algorithm and one about testing automated steering control. These studies are used to validate the results of this study.

3.1 Automated Robustness Testing with UAV

In this previous study, three test generation methods were compared regarding the performance of robustness by finding the most challenging test cases for a software (Betts and Petty, 2016). Two different kinds of Search-Based Testing methods were used to generate the most challenging test cases for an autonomous unmanned aerial vehicle (UAV) model in a closed-loop simulation. These Search-Based Testing methods were Genetic Algorithms and Surrogate-based Optimization, which were later compared against Monte Carlo with a uniform probability distribution and full combinatorial testing. The Search Based Testing methods outperformed the other testing methods when it came to "finding the most challenging test case and finding the set of fifty test cases with the highest mean degree of challenge".

The performance of the three automated testing algorithms was evaluated by setting the total numbers of allowed simulation executions for each experimental trial. Furthermore, two metrics were used to evaluate the performance of generating the most challenging test cases (Betts and Petty, 2016). The first metric was based on maximum lateral deviation for one single test case and the second metric was based on the highest mean lateral deviation for 50 test cases.

In the first metric, i.e. the maximum lateral deviation for one single test case, Monte Carlo outperforms the Genetic algorithm in generating the most challenging test case for low execution runs. However, when reaching 200 simulation executions, the Genetic algorithm starts to outperform Monte Carlo. A Genetic algorithm requires a balance between a number of generations and population size in order to improve its result. It initiates a random population size of test cases, sends the test cases for simulation and then evaluates the deviation (Betts and Petty, 2016). Monte Carlos input and output is always random, which means it does not evaluate the deviation value (Thomopoulos, 2013). Therefore, it makes sense that the Genetic algorithm outperforms Monte Carlo when the number of simulation executions increases (Betts and Petty, 2016).

The Surrogate-based optimization algorithm (SBO), slightly outperforms Monte Carlo when running 50 simulation executions. However, as the simulation execution runs increases, SBO significantly outperforms Monte Carlo (Betts and Petty, 2016). Furthermore, SBO outperforms the Genetic algorithms for all trials and is able to find the most challenging test case when the simulation execution runs are 1000 and 2000. The most challenging test case having a lateral deviation value of 7.47 m, which is the true maximum.

In the second metric, i.e. mean lateral deviation for fifty test cases, Monte Carlo outperforms the Genetic algorithm, which is the same as for maximum lateral deviation (Betts and Petty, 2016). The Genetic algorithm outperforms Monte Carlos at the same value of simulation execution runs as mentioned before. In addition, the Surrogate-based optimization algorithm (SBO) outperforms the Genetic algorithm for every simulation execution runs, except for 2000 runs. At 2000 simulation execution runs, the SBO algorithm reaches a lower mean value than the Genetic algorithm (6.88 m for the SBO and 6.92 m for the Genetic algorithm). The mean values indicate that neither of the algorithms reached the true maximum, which was 7.47 m.

3.2 Robust Autonomous Vehicle Controller

According to Schultz, Grefenstette and De Long (1993), verification and validation are inadequate when testing the robustness of an autonomous vehicle controller. The controller might not operate as stated, due to various situations with different condition parameters. Testing all situations would not be feasible since it would require isolated environments and intense labor, which is costly and time-consuming. By creating a vehicle simulation environment with different fault scenarios and the use of test generation methods, these issues can be partially solved. In this previous study, a vehicle simulation environment based on a UAV model controller was created, together with a minimal set of faults that contains degraded vehicle performance (e.g. sensor faults, control faults) and a maximum set which consists of acceptable faults. In addition, a genetic algorithm was used for searching after fault scenarios of interest (see Figure 3.1).

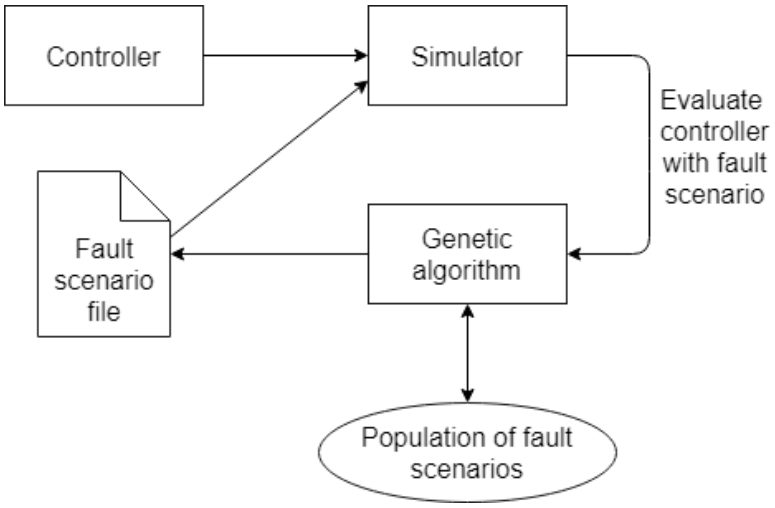

Figure 3.1: Robustness testing of an autonomous vehicle controller using a genetic algorithm (Schultz, A.C., Grefenstette, J.J, De Long, K.A., 1993).

The genetic algorithm was executed for 100 generations with a population size of 100, i.e. containing fault scenarios, which gave 10,000 evaluations. Findings regarding the robustness of the UAV model's controller showed that sensor faults were severe to recover from and that control faults were easier to handle. It was easier to recover from faults that alter the UAV model's air pitch compared to faults altering its roll. In addition, some sensor faults erased other control faults, which gave a positive effect (Schultz, Grefenstette and De Long, 1993). The findings from this paper conclude that the UAV model controller needs to be enhanced in order to increase its robustness.

3.3 Random Testing for Dynamic Test Data Generation

The goal of software testing is to find test inputs that trigger particular features and reveal as many faults, e.g. software errors that causes a system to malfunction, as possible (Michael, et al, 2002). Manual testing is not feasible in achieving these tasks since it is costly and time-consuming. Therefore in this previous research, random test generation and a genetic algorithm were compared on how efficient they could solve these problems. In the experiment condition decision coverage (CDC) was used as test criterion and for benchmarks miniature programs were used, e.g. binary search and triangle classification. The findings in this previous article show that random test generation was unable to accomplish the task in achieving a tolerable result for the test criterion, i.e. finding inputs for CDC. Meanwhile, the genetic algorithm was more superior in meeting the test criterion, e.g. it achieved a higher percentage coverage during testing of the programs mentioned above. The findings drawn from this article showed that random test generation was outperformed by the genetic algorithm in finding test inputs that achieved the mentioned test criterion.

This means that random test generation could be less efficient than the genetic algorithm regarding other test criteria, e.g testing robustness. As mentioned by Lei, et al (2010) robustness is defined as *how well the software performs under stressful environmental conditions or how well the software handles invalid insertions*, which means that random test generation could be less efficient in finding suitable inputs for testing the robustness of software systems.

3.4 Autonomous Steering for Lane Keeping

In the last years, the importance of having a robust autonomous steering control system has grown rapidly, for example driving safety and energy savings (Chu, et al, 2018). According to Filho, et al (2014), it is not easy to design a simple autonomous steering control system since it is difficult to select and measure various kinds of parameters, e.g. uncertainties and disturbances. Uncertainties in a simplified model arise when lateral and longitudinal dynamics are not considered. The autonomous vehicle needs to handle this by having longitudinal and lateral controllers, that assists in path following and cruise velocity. Other uncertainties e.g. roll and heavy motions, are often left out in a simplified model and assuming small steering and slip angels on the wheels (Chu, et al, 2018), which can cause serious consequences.

Besides uncertainties, autonomous vehicles need to handle different kinds of disturbances, for example, different road conditions, wind forces, and road adhesion. Chu, et al (2018) argues that it is important that a simple autonomous steering control can regulate these types of disturbances before being implemented into a real vehicle. In addition, it is desirable to implement a simple autonomous steering control since it is easier to implement and requires less real-time measurements and calculations. As long as the model achieves the conditions mentioned above, is robust and improves the system reliability, it is preferable over a complex autonomous steering control.

There exist some available options for constructing a robust autonomous steering control system, e.g. active disturbance rejection control (ADRC) and proportional integral derivative (PID) steering control. The ADRC consists of three parts, these are ESO, TPG, and NWS. ESO stands for extended state observer and estimates in real-time internal uncertainties and external disturbances. NWS stands for nonlinear weighed sum and rejects uncertainties and disturbances before they infect the system. TPG stands for transient profile generator and generates a profile which the system can follow (Chu, et al, 2018). Meanwhile, the PID steering controller regulates the vehicle dynamics, i.e. the steering wheel angle, using the yaw rate and by receiving lateral deviation measurements from an established distance point (Marino, Scalzi, and Netto, 2011).

The autonomous steering control systems were validated against different parameters. These included e.g. mass, wind force, magic formula, longitudinal velocity, lateral velocity and steering angles. The parameters were altered in order to check the stability and deviation of the autonomous steering control systems. According

to Nowak, Czeczot, and Klopot (2018), the ADRC has shown a better performance improvement in the recent years for both simulations and experimental tests, i.e. regarding lane keeping, in comparison with the PID steering controller. However, both of them are still facing major challenges before one could determine the best solution for lane keeping.

4
Research Design

This chapter introduces the research questions for this study and the research method, as well the realization of design. It discusses how the research questions will be answered by using the selected research method.

4.1 Research Questions

The following research question RQ1 and RQ2 have been produced for addressing the problem of which of the two test generation method, SBT with a genetic algorithm or MC with a continuous uniform probability, that most efficiently tests the robustness of the autonomous vehicle model in a simulation environment by inserting tests cases and manipulating the parameters in these test cases. In addition, RQ3 have been produced in order to guarantee that the vehicle model does not affect the result from the two test generation methods.

RQ1: **Which method is more efficient in testing the robustness of the autonomous vehicle model, Monte Carlo or genetic algorithm, by inserting test cases in a scenario and manipulating the test case values, i.e., the condition, road surface, and the fault, steering miss alignment, as well as their coordinates ?**

Robustness: is defined as how well the software performs under stressful environmental conditions or how well the software handles invalid insertions (see section 2.1).

RQ2: **How well do the test generation methods perform compared to each other ?**

- Hypothesis 1: Search based testing should outperform Monte Carlo's testing in finding the most challenging test cases.

- Hypothesis 2: Search based testing performance should increase when computational runs are increased and Monte Carlo's testing performance should decrease.

RQ3: **How can we guarantee that the autonomous vehicle model doesn't affect the data result from the test generation methods ?**

- Hypothesis 3: When applying test cases without any conditions and faults, the autonomous vehicle model should stay on the specified path in the scenario.

4.2 Research Approach

This section explains the selected research approach and explains how to conduct it, which phases are included etc. Lastly, it describes how the research approach was applied in the study in order to answer the research questions.

Selected Research Approach

The research questions RQ1, RQ2, and RQ3 were answered by a constructive research approach, which is shown in Figure 4.2 (Piirainen and Gonzalez, 2013). This was found as the most suitable approach for this study since a solution for a practical industry problem was constructed and validated with theory. The research approach is discussed in more detail in the next section.

Apply Constructive Research

A constructive research approach is a methodology which contributes to the field of science by constructing an innovative solution (Piirainen and Gonzalez, 2013). It consists of different phases which need to be followed according to the approach (see figure 4.2):

1. "Finding a practical relevant problem that has a research potential" (Oyegoke, 2011) which needs to be solved in real-life and that interests practitioners (Piirainen and Gonzalez, 2013). A practical problem can be found in different areas, for example, the industry, where the practical problem could be related to customer dissatisfaction, quality standards etc. When a practical problem has been identified it needs to be substantiated with literature, which is done by "in-depth literature review" (Oyegoke, 2011). After substantiating the practical problem with literature, the proposition needs to be tested in order to find gaps which have not been covered.

A major problem in today's car industry is that autonomous vehicles are facing major challenges when it comes to testing vehicles towards different scenarios. Today this is solved by different simulations using various autonomous testing methods (see the statement of problem section). However, this has not been achieved in the Revere Lab at Charmers University. Therefore, the knowledge gap of this study is to develop a simulation for an autonomous vehicle model and to test two different test generation methods, Monte Carlos and Search Based testing (see research questions). The testing results from these methods will be used to refine theories regarding performance and robustness (see hypothesis one and two).

2. "Obtaining a general, comprehensive understanding of the topic" (Oyegoke, 2011) by analyzing the problem and reviewing literature (Piirainen and Gonzalez, 2013). Literature is collected and analyzed in order to get a thorough understanding of what has been achieved so far from the topic (Oyegoke, 2011). Several innovative and incentive ways should be documented and examined how the problem can be solved. A comparison between the solutions should be carried out in order to achieve a better understanding (Oyegoke, 2011).

In order to solve the problem in this study and answer the research questions, literature will be selected and serve as the base for this research. Literature will be analyzed in order to achieve a higher quality background. Furthermore, previous theories and results (see related work) will be used for validating the research results, as well as validating the correctness of the simulation.

3. "Innovating – designing a new construct" (Oyegoke, 2011) which means according to (Piirainen and Gonzalez, 2013) proposing and constructing an innovative solution to the practical problem. The solution should be based on practical experience or theory from the previous phases (Piirainen and Gonzalez, 2013). The knowledge should be used to guide the researcher in understanding the targeted phenomenon (Oyegoke, 2011).

The problem will be solved by constructing an innovative solution, i.e. a simulation environment for testing the robustness of an autonomous vehicle model using two test generation methods, which will be based on previous theories.

In addition, the results from the solution will be used to answer the research questions. RQ1 and RQ2 will be answered with the results achieved after completing the following steps:

- Initiate a scenario case which consists of coordinates. The vehicle model needs to follow these coordinates in order to stay on the path.

- Run a base test without any faults in order to validate that the vehicle follows the designed path.

- Create test cases based on the scenario with one fault, steering miss alignment, and one condition, road surface, for each test case (see Figure 4.1). The random fault and condition will receive random coordinates in each test case.

- The test cases are then sent to the simulation for evaluation. Each test case will be executed once in the simulation and will receive an id and a deviation value. The deviation value is calculated by using Dynamic Time Wrapping (DTW), which is further discussed in section 4.3.

- The test generation methods will then receive the test cases and will perform their operations. Furthermore, new test cases will be generated which are sent

to the simulation if criterion is not met, i.e. limit the number of runs.

- When criterion is met, the loop will terminate and the mean deviation value will be calculated for the test cases (see formula below). The greater the mean deviation value is, the better the automated testing algorithms are at making the autonomous vehicle model less robust.

  Mean deviation formula:

  The deviation value = $f(d_n)$ is calculated by using DTW for each test case, whereas n is the index for the first executed test case. The mean deviation is then calculated by using the generated values from DTW and using the formula listed below (see section 4.3 for an explanation of how DTW operates).

  Mean deviation = $\frac{1}{t}\sum_{n}^{t} f(d_n)$, were t = number of test cases and n=1.

  Example:

  Lets say that DTW calculated the deviation values $f(d_1) = 5$ m and $f(d_2) = 4$ m for two test cases, then the mean deviation is calculated by:

  Mean deviation = $\frac{1}{2}\sum_{1}^{2} f(d_1) + f(d_2) = \frac{1}{2}(5+4) = 4.5$ m

- The evaluation is then based on the metric: which automated testing method found the most challenging test cases that generated the greatest mean deviation value.

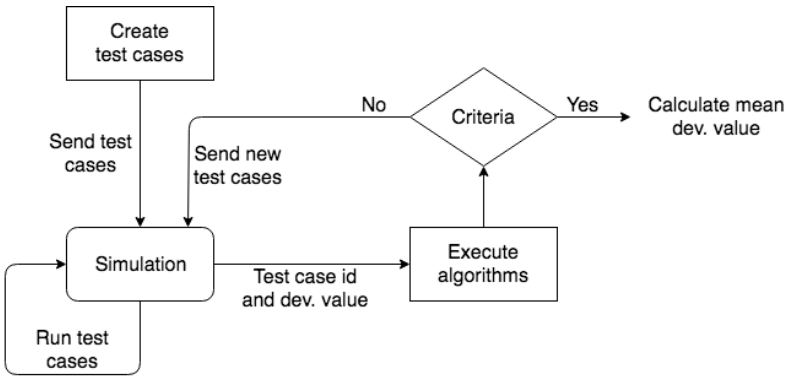

Figure 4.1: Shows the process of how RQ's will be answered.

Supplementary to the process above, additional tests will be performed using the same scenario with the exception that no faults and conditions will be inserted. This is to ensure the precision and consistency of the developed simulation and to further support RQ3.

4. "Demonstrating that the new construct (solution) works" (Oyegoke, 2011) and implementing the artifact (Piirainen and Gonzalez, 2013). The validation can be done in several ways. A pilot case study would be the most proper way to test and improve a construct. However, depending on the industry it could be risky and costly. In this case, different triangulation based approaches should be used. There are four different triangulation types. The first one is called "data source triangulation" which means that the data will remain the same in different situations. The second one "investigator triangulation" is when several investigators examine the same event. The third one is called "theory triangulation" which means several investigators with different views examines the same result. The fourth one "methodological triangulation" is when several approaches are used in order to gain more evidence, which increases certainty in the concept (Oyegoke, 2011).

Furthermore, when pilot projects cannot be used, analogical and theoretical validation can be an alternative (Oyegoke, 2011). Analogical validation is when existing project solutions are compared with the proposed solution. This is done by reviewing existing project case survey examples (Oyegoke, 2011). Theoretical validation is another approach of comparing the proposed solution with theory. In this case, it is usually a general theory, for example, organizational theories (Oyegoke, 2011). Furthermore, empirical validation can be used to validate the proposed solution. This approach uses interviews and questionnaires to retrieve the perceptions of people who are involved in similar project cases (Oyegoke, 2011).

In this study, a scientific approach will be used for validation in order to compare and validate the collected theory, i.e. background, and related work, with the research results. The theory will be used together with the research result to test and verify the hypothesis from the research questions, as well as answer the RQ's. This is what is known as a "Deductive theory" approach, which means general theory predictions are verified with particular research results (Johnson, 1996). In this case, the research results will verify if the hypothesis is true or false in this study. In addition, an empirical validation will be used to validate the autonomous vehicle model and answering RQ3 by running the model in various frequencies with 50 test cases that contain no faults and conditions.

5. Displaying the research contribution of the solution to the problem and theoretical connections (Oyegoke, 2011; Piirainen and Gonzalez, 2013). This is done by establishing steps including research procedures and operational measures, which needs to be documented throughout the study. The reliability of these steps can later be reviewed by other practitioners (Oyegoke, 2011). Furthermore, the ex-ante knowledge proposition which motivates the artifact of conducting the study needs to be compared with the results of the study (Piirainen and Gonzalez, 2013). This shows the theoretical connections and contribution of the study by displaying how the outcome refines existing theories or by establishing a new theory (Piirainen and Gonzalez, 2013).

The research contribution of the solution to the problem is done by demonstrating the importance of test generation methods for validating the robustness of autonomous vehicle models. A performance comparison between the two test generation algorithms contributes to the field of research by displaying that the genetic algorithm should outperform Monte Carlo concerning robustness when comparing them using the innovative solution, i.e. the simulation environment. The results from the solution will be used to refine previous theories regarding robustness performance of the two test generation methods.

6. "Examine the scope of applicability of the solution" (Oyegoke, 2011). The applicability of the solution is shown by demonstrating that a solution which works for one case should work for similar cases (Piirainen and Gonzalez, 2013). This claim is possible after linking theory to the solution and analyzing the problem, which should be already done after completing the previous phases (Piirainen and Gonzalez, 2013).

The applicability will be examined by comparing the solution results with previous research results. If the result from the two test generation methods follows the same pattern as the previous research solutions, then a similar simulation environment should receive the same result patterns, after using the test generation methods. In addition, the solution will be constructed in such a way that various autonomous vehicle models can be tested.

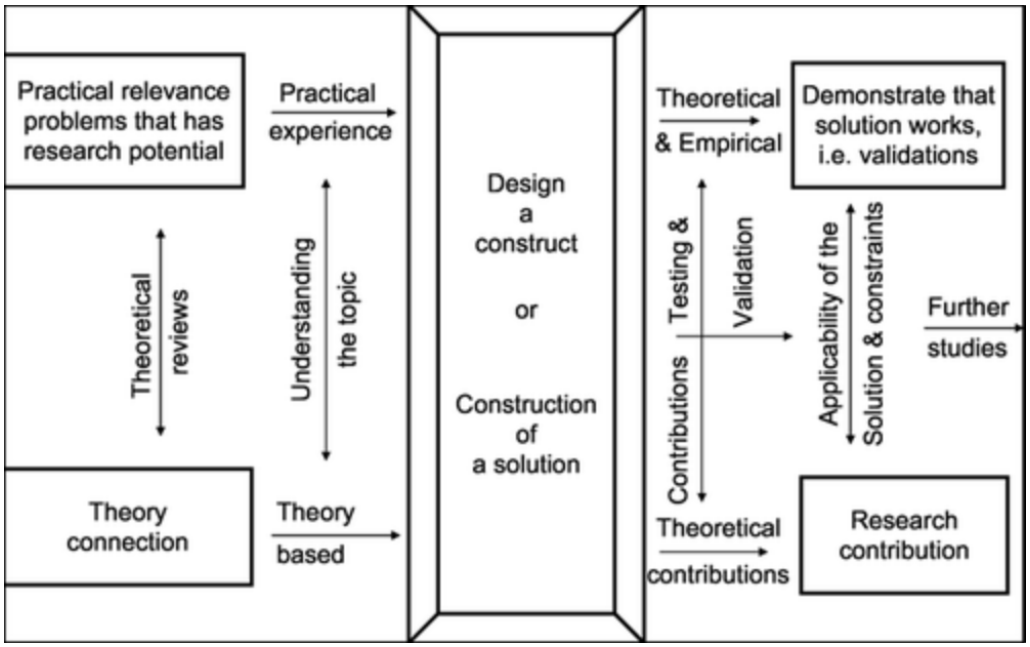

Figure 4.2: The constructive research process, step by step (Oyegoke, 2011).

4.3 Realization of Design

The developed software for this study is an event based test environment using the racing game Vdrift and OpenDaVinchi. The overall architecture is shown in Figure 4.3.

The simulation system will be modular based where custom faults and conditions, vehicle models and scenarios can be used. The vehicle model supplied for this study is updated with a specific frequency and will not be based on events like the rest of the system. In the simulation environment, one will generate a scenario, add test cases and the total amount of simulation runs, as well type of algorithm and other parameters. The generated test cases will be parsed by the simulation test suit, which will generate the number of test cases and runs specified. While the simulation is starting and loading. The simulation receives input via events from a start and end scenario, as well as faults and conditions. The data from the vehicle model are later collected and sent to the test suit for parsing and for generating new test cases.

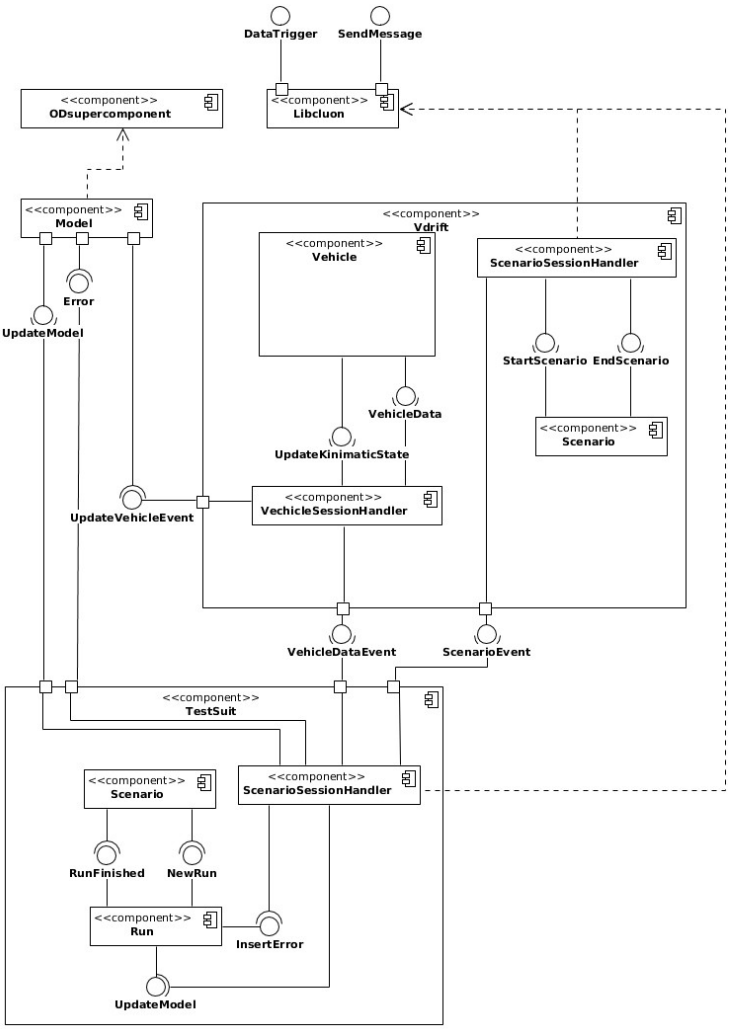

Figure 4.3: Overall architecture

The scenario developed for this study simulates lane changing and is visualized below in figure 4.4. The car is designed to accelerate to 20 km/h, make a lane change and keep the new lane. The total length of the scenario is 190 meters. The yellow path indicates the path the vehicle should keep during the scenario whereas the red one illustrates the deviated path.

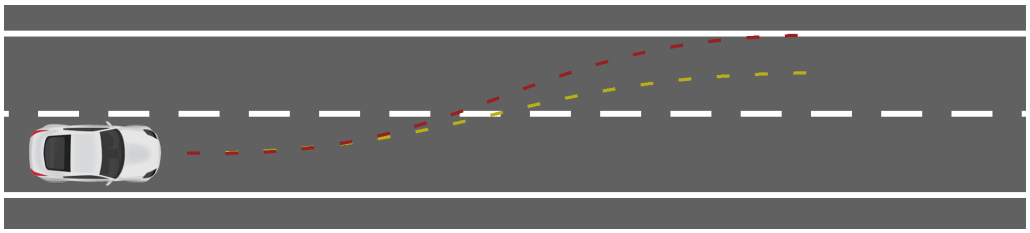

Figure 4.4: Visualization of scenario

During the execution of the scenario, one of the two errors is inserted, respectively a reduction of friction which represents a road surface or a steering miss alignment. These can be inserted at any time, and for this scenario, it can occur at one of the twelve specified coordinate points. The deviation is then calculated by comparing the two paths using Dynamic Time Warping (DTW). DTW finds the most optimal match between the two paths and takes stretching and compression of the path into account, which results in that speed and path changes are taken into account during the scenario. DTW is visualized below in figure 4.5.

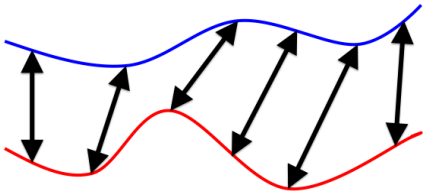

Figure 4.5: Dynamic Time Warping

The visual representation of the scenario is illustrated in figure 4.6 below. It is based on a default race track within VDrift. The vehicle in the figure has just made the lane change and will soon complete the scenario. Only a portion of the long straight track is used.

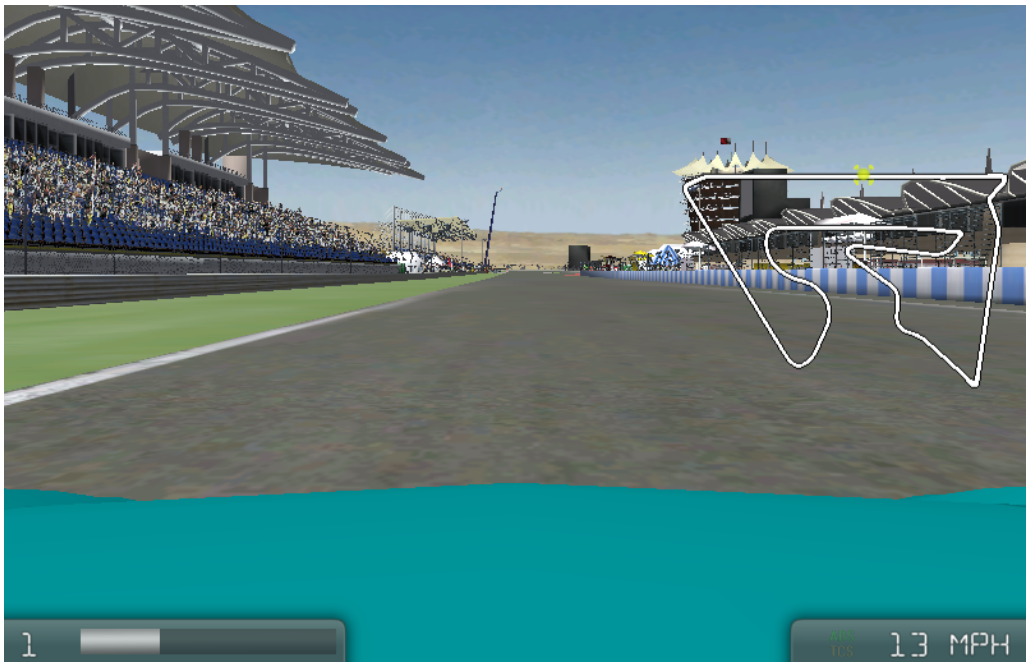

Figure 4.6: The visualization of the simulator

5
Results and Discussion

This chapter includes an analysis of the study results (see appendix for the actual results) which are used for answering the research questions. It also includes a discussion of the study results, as well a discussion about threats of validity, and sustainability.

5.1 Answering Research Questions

In order to be able to answer the research questions, several steps had to be made beforehand. Related literature had to be analyzed and a simulation environment had been developed for testing the autonomous vehicle model. In addition, two automated test generation methods had to be implemented (search based testing with a genetic algorithm and Monte Carlo with a continuous uniform probability distribution). After these steps were completed, results could be generated to answer the research questions.

Finding the Most Challenging Test Cases

The two test generation methods, Monte Carlo and Search-based testing with a genetic algorithm, were implemented to find the most challenging test cases, which contains the fault, steering miss alignment, and the condition, road surface, for 4 trials. For each simulation execution in the trials, 10 test cases were used. Table 5.1 shows the highest deviation value by the number of runs for the two automated algorithms. As the table shows, the genetic algorithm significantly outperformed Monte Carlo at finding the most challenging test case after 10 runs. However, at 20 runs, Monte Carlo's performance starts to align with the genetic algorithms and continues to do so. When completing 80 runs, the genetic algorithm and Monte Carlo almost finds the same maximum deviation value (8.41 m for the genetic algorithm and 7.78 m for Monte Carlos). This makes sense since Monte Carlo's probability of randomly finding the most challenging test case increases when the runs increments.

Trial	1	2	3	4
Simulation executions	10	20	40	80
Monte Carlos	1.6474	6.9360	7.5316	7.7787
Genetic algorithm	5.3687	7.5626	7.7537	8.4084

Table 5.1: Shows highest deviation value of the two automated algorithms by number of runs.

Table 5.2 shows the mean deviation value for the 10 most challenging test cases. In each trial, the simulation executions increased by a factor of two. Starting at 10 simulation execution runs for each algorithm and then to 20, etc. The mean deviation value is based on one fault, steering miss alignment, and one condition, road surface which was calculated by using DTW as mentioned earlier for comparing the different paths after the simulation execution runs were completed. In this case, the algorithm which generated the 10 most challenging test cases and that gave the largest mean deviation was the genetic algorithm. The genetic algorithm significantly outperformed Monte Carlos for the first 10 runs and continues to do so. This is expected since Monte Carlo will randomly explore more unfavorable solutions when the test suite increases compared to the previous metric (Betts and Petty, 2016). Still if the simulation runs continue to increase, Monte Carlo will eventually align with the genetic algorithm's performance, however, this is more time-consuming and costly. The result from Table 5.1 above and Table 5.2 below will be used to answer Research Question 1 and 2.

Trial	1	2	3	4
Simulation executions	10	20	40	80
Monte Carlos	0.5435	0.6603	0.8501	1.1464
Genetic algorithm	2.3967	2.7822	4.2262	4.5151

Table 5.2: Mean deviation value of the two automated algorithms.

In addition, Figure 5.1 will also be used for answering Research Question 1 and 2. It illustrates the automated test generation methods mean deviation per run for the total number of runs, where the genetic algorithm rapidly discovered the most challenging test cases. The results show that the Monte Carlo will eventually converge with the generic algorithm.

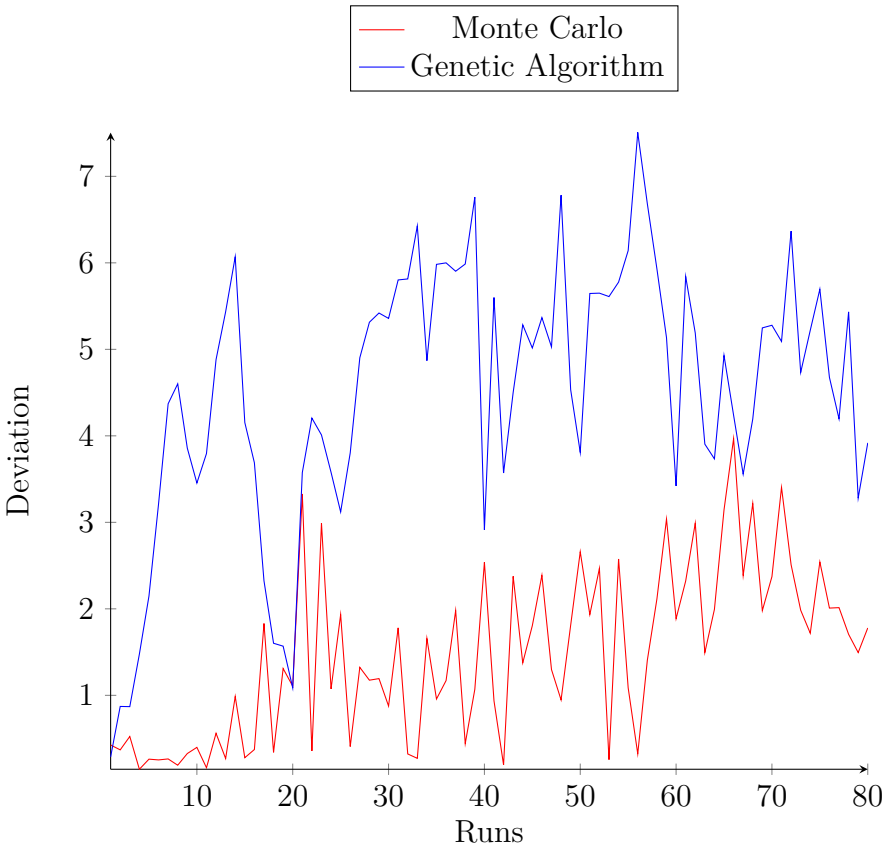

Figure 5.1: Mean deviation for each run.

Research Question 1

To answer the research question *"Which method is more efficient in testing the robustness of the autonomous vehicle model, Monte Carlo or genetic algorithm, by inserting test cases in a scenario and manipulating the test case values, i.e., the condition, road surface, and the fault, steering miss alignment, as well as their coordinates ?"* study results needs to be discussed with corresponding literature.

The study results shows that the genetic algorithm is the most efficient method for testing robustness of the autonomous vehicle model. The genetic algorithm was able to produce better test cases than Monte Carlos for testing robustness since it generated larger deviation values for the autonomous vehicle model.

Examining Figure 5.2 shows that the genetic algorithm was able to produce a larger maximum deviation than Monte Carlos after 80 simulation runs. None of them reached the true maximum, but the graphs in Figure 5.2 displays that the genetic algorithm will reach it faster. The margin between Monte Carlo and the genetic algorithms maximum deviation is small, which is shown in Table 5.1 (8.41 m for the genetic algorithm and 7.78 m for Monte Carlos). This makes sense since only one test case was used, which means that the genetic algorithm did not use its global search efficiently (Harman, et al 2010), due to the low test case population. In the case of Monte Carlos, the input and output are always random when using a continuous uniform probability distribution (Thomopoulos, 2013), which means that it explores unfavorable solutions. Although, due to the small test case population, it explored a quantity number of unfavorable solutions.

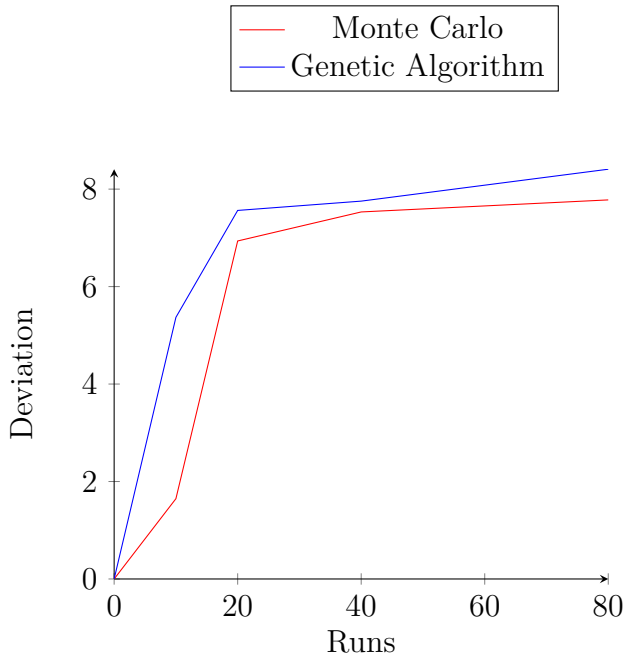

Figure 5.2: Maximum deviation for testing methods.

Meanwhile, analyzing Figure 5.1 and Figure 5.3 displays that the genetic algorithm

significantly outperformed Monte Carlos in discovering the largest mean deviation for 10 test cases after 80 runs. Table 5.2 exhibit that the mean deviation value for 80 runs were 4.51 m for the genetic algorithm and 1.15 m for Monte Carlos. The reason for this lies in how Monte Carlo's algorithm operates. As stated by Thomopoulos (2013), Monte Carlo's input and output are always random when using a continuous uniform probability distribution. This means that Monte Carlo will explore more unfavorable deviation values as the test case population increases. However, these unfavorable deviation values could be of interest in some cases, which is why Monte Carlo is popular today. As stated by Michael, et al, (2002) *"The goal of software testing is to find test inputs that trigger particular features and reveal as many faults, e.g. software errors that causes a system to malfunction, as possible"*, and since the location of these faults is unknown, random algorithms could be preferred.

In addition, the performance of the genetic algorithm increases with the augmentation of the number of test cases. The reason for this is mentioned in the article by Harman, et al (2010), which explains that the genetic algorithm performs a global search with a fitness function, which makes it possible for the algorithm to find the best test cases during each evaluation. By using its operations, the genetic algorithm is able to learn and keep track of which test case that gave the largest deviation value for each evaluation. This gives the genetic algorithm an advantage over Monte Carlos, as it is not based on a random search.

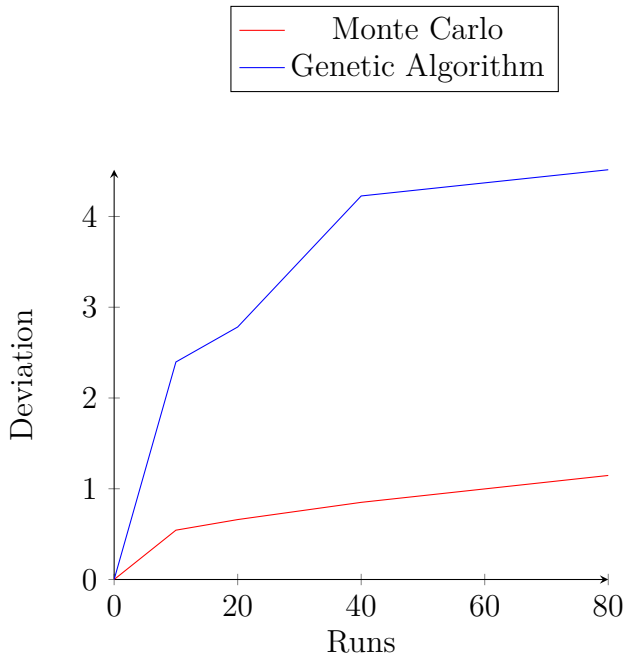

Figure 5.3: Mean deviation for testing methods.

As previously mentioned, the findings demonstrate that the random algorithm, i.e. Monte Carlos, has difficulties in finding the best test cases to meet the test criteria (largest deviation), which matches previous theory in Michael, et al (2002). However, the validation of the research question regarding robustness can only be based

on the result of one and ten test cases. If the test case population would be increased to e.g. 500 or 1000 test cases, the results and answer could vary.

Research Question 2

The second research question, i.e. *"How well do the test generation methods perform compared to each other ?"* is answered by using the associated hypotheses. In order to present a proper answer to the research question, the hypotheses are validated by using the study results and equivalent literature.

The first hypothesis states that *"Search-based testing should outperform Monte Carlo's testing in finding the most challenging test cases."*, which became a true statement according to the study results. Considering the results of looking for the most challenging test case that gave the highest deviation value for the autonomous vehicle model. Figure 5.2 displays that the search based testing method with a genetic algorithm outperformed Monte Carlos in finding the test case that produced the largest deviation, especially for low simulation runs.

The second hypothesis mentions that *"Search-based testing performance should increase when computational runs are increased and Monte Carlo's testing performance should decrease."*, which in this study argues that it turned into both true and false depending on the test case metric. The finding shows that this clearly is this case when examining the results from the mean deviation measurements (see Table 5.2).

The search based testing method with a genetic algorithm significantly outperformed Monte Carlos in finding the ten most challenging test cases. Examining the mean deviation values in Table 5.2, one can clearly see that the genetic algorithm's performance increased as the computational runs increments, which is not the case when analyzing Monte Carlo's performance. Monte Carlo was able to find larger deviation values than its previous, but it didn't increase as much rapidly as the genetic algorithm (see Figure 5.3).

However, this doesn't apply when finding the most challenging test case. When examining table 5.1 the hypothesis becomes false, since Monte Carlo's performance also increases rapidly and almost reached the same values as the genetic algorithm. The logic behind this could be related to as previously mentioned by Thomopoulos (2013), that Monte Carlos with a continuous uniform probability distribution should explore more unfavorable solutions but in this case only discovered a portion of these, due to a small test case population.

As the graph show in Figure 5.1, Monte Carlo will eventually align with the genetic algorithm in finding the ten most challenging test cases. However, this would require additional simulation runs, which will be time-consuming and increase testing cost according to Michael, et al, 2002; Schultz, Grefenstette and De Long, 1993; Bogdan,

1990.

The answer to the research question drawn from the hypotheses is that the search based method outperformed Monte Carlos significantly in finding the ten most challenging test cases. Still, when comparing them against the metric of finding the most challenging test case. The genetic algorithm outperformed Monte Carlos but with a small margin and not as rapidly as mentioned above. In addition, the validation of the answer can only be based on testing one and ten cases, since the performance can be different when testing larger test case populations.

Research Question 3

In order to answer the research question *"How can we guarantee that the autonomous vehicle model doesn't affect the data result from the test generation methods ?"* a comparison is required with previous solutions, as well with an empirical validation (Oyegoke, 2011), not only to achieve a better understanding but also to ensure the accuracy and correctness of the simulation as well as of the result.

The empirical validation was made using the same scenario and by running a total of 50 additional tests which did not contain any faults and conditions, where 10 test cases were launched with various frequencies (1 Hz to 40 Hz), with which 5 runs were executed. This was done in order to find incorrectness of the vehicle model and the simulation, as well answer the third hypothesis *"When applying test cases without any conditions and faults, the autonomous vehicle model should stay on the specified path in the scenario."* This resulted in testing the vehicle model with different update frequencies. Table 5.3 and Figure 5.4 below show that the higher the frequency was, the more accurately the simulation result would be. The validation results demonstrate that the deviation within the same scenario was statistical insignificance with less than 1% overall deviation with 40hz when comparing it with the scenario path. Thus the vehicle stayed on the specified path, which means that accuracy of the simulation environment is adequate for testing the performance of the two test generation methods and that the model's behavior does not affect the test results. If the deviation values would have been larger e.g. 0.5 m, the data collected from the original test runs could be a result of an inaccuracy rather than the algorithm itself. This only partially verifies the accuracy of the collected test result data from the scenario. Other non-tested scenarios could be different and the deviation could result in more than 1%. The same can be discussed regarding having a longer scenario.

Frequency	1	5	10	20	40
Deviation	0.1013	0.0499	0.0393	0.0115	0.0107

Table 5.3: Deviation in cases of different frequencies.

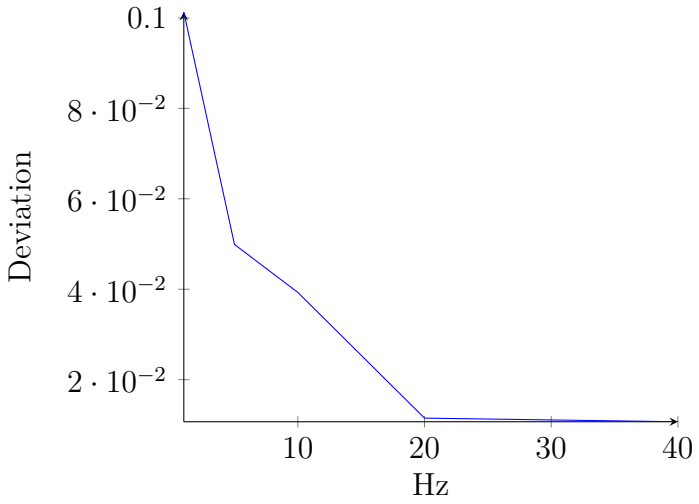

Figure 5.4: Mean deviation for different frequencies.

In order to validate the generated research results from the test generation methods, a comparison with previous research results was made. Figure 5.5 and 5.6 represents the result from Table 5.1 and 5.2 in comparison to Table 2 from Betts and Petty (2016), as well Table 4 from Betts and Petty (2016), in order to verify the correctness of the developed simulation environment. In Betts and Petty (2016) the algorithms were executed for a total of 2000 runs, while in this study they were executed for 80 runs. The reason for only using 80 runs in the study was that the algorithms were executed in a simulation environment with scenario visualization, which was more time-consuming than the simulation used in Betts and Petty (2016).

The first comparison shows that after a limited amount of runs, the genetic algorithms are able to reach a greater maximum deviation value than the Monte Carlos, shown in Figure 5.5. In the case of Betts and Petty, (2016), the largest deviation value for one test case was found after 2000 runs, and in this study, it was found after 80 runs. However, none of them reached the true maximum deviation since the graphs are still increasing. A comparison between the graphs pattern in this study and the previous one shows that both the genetic algorithm and Monte Carlo are correctly implemented.

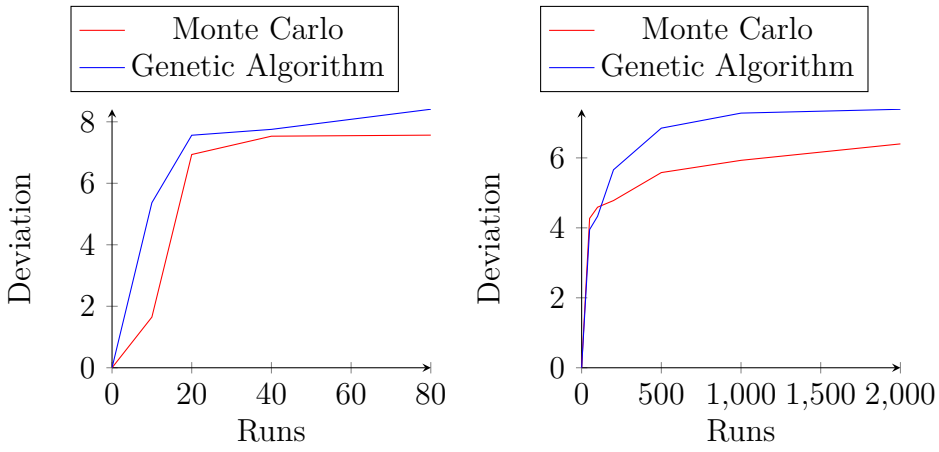

Figure 5.5: Left: Table 5.1 of Maximum deviation, Right: Table 2 from Betts and Petty (2016).

The second comparison displays that the genetic algorithm significantly outperformed Monte Carlos at finding the mean deviation. Still, this is not the case in Betts and Petty (2016) when looking at lower than 200 runs. In this study, the genetic algorithm greatly outperformed Monte Carlo's from 10 runs to 80 runs.

Meanwhile, in Betts and Petty (2016), this didn't occur until the testing reached greater than 200 runs. According to Betts and Petty (2016), the genetic algorithm requires a balance between a number of generations and population size in order to improve its result. The reason for the divergence could be that the genetic algorithm achieved this balance earlier, since only one fault and one condition were used, compared to the previous study which used three faults and six initial conditions. In addition, the fault and condition parameters used in Betts and Petty (2016) are not the same type as the ones used in this study, which means that they can affect the deviation differently. Furthermore, the conditions used in Betty and Petty (2016) are not manipulated, which can be another reason for the divergence. However, by examining the simulation accuracy in Figure 5.4, the overall mean deviation for each run in Figure 5.1, as well the graphs pattern for high simulation runs, indicates that the genetic algorithm and Monte Carlo were correctly implemented.

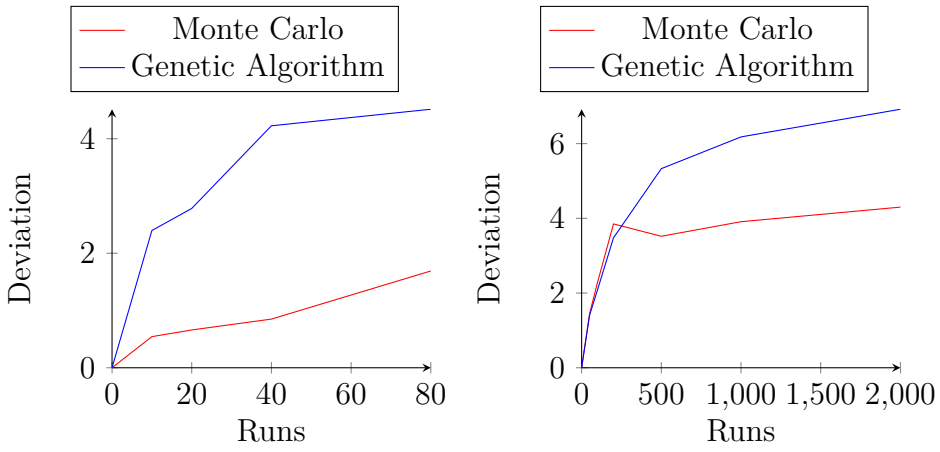

Figure 5.6: Left: Table 5.2 of Mean deviation, Right: Table 4 from Betts and Petty (2016).

The empirical validation and comparison with previous literature results validate the following conclusion that the simulation environment and the vehicle model are accurate for applying and testing the two test generation methods regarding robustness.

5.2 Threats to Validity

The factors which could threaten the results and validation of this study were that the autonomous vehicle model was updated with a specific frequency. When validating the simulation environment by running 50 test cases without any faults and conditions, one could see that the simulation environment and the vehicle model was accurate enough for validating the results (see table 5.3). However, the autonomous vehicle model was quite simple which means if a more complex vehicle model such as the model presented in Gorelov, Komissarov, and Miroshnichenko (2015) would be implemented, it would require more resources. This, in turn, could affect the validation of the research results in this study.

Time threatens the results and validation of this research as the results could vary for greater simulation runs. Inserting more faults and conditions could also have affected the results and look more like the pattern in Betts and Petty (2016) for low simulation runs.

The previous results by analyzed literature could affect the validation of this study since we can't guarantee that they implemented the test generation methods correctly.

5.3 Sustainable Autonomous Vehicle Testing

According to NHTSA (2018), 93 percent of car accidents between 2005 to 2007 was due to human factors. If autonomous vehicles could prevent these human errors, then many lives could be saved (Greenblatt and Shaheen, 2015). As stated by Berger (2014): to able to replace the driver and ensure safety, the autonomous vehicle needs to handle all various kinds of scenarios that it can be exposed to. Otherwise, it could make the same mistakes as the driver or even worse decisions.

By testing the autonomous vehicle systems against various scenarios using a simulation environment and test generation methods, this can be solved. The solution makes it possible to test the autonomous vehicle systems by repeatedly exposing it to different kinds of scenarios with various faults and conditions, which makes it more robust and reduces its chances of making the same mistakes as a human driver. It also reduces the CO2 emissions since a simulation environment doesn't require a testing ground. This makes sense since a scenario e.g. parking a vehicle, which could produce up to 1.5 to 25 grams of CO2 according to Greenblatt and Shaheen (2015), can instead be tested using a simulation environment. The findings mentioned above shows that simulation testing can have a huge impact on sustainability when testing autonomous vehicles.

6
Conclusion

In this study, the problem of testing an autonomous vehicle towards different kinds of scenarios with various parameters was solved, by creating a simulation environment for an autonomous vehicle model, analyzing literature, and by implementing two test generation methods. The two test generation methods, Search Based testing with a genetic algorithm and Monte Carlos, were compared in how well they could increase the robustness of the autonomous vehicle model. The genetic algorithm was able to significantly outperform Monte Carlos in finding the ten most challenging test cases since Monte Carlos explored more unfavorable solutions, due to random input and output. When it came to finding the most challenging test case, the genetic algorithm clearly outperformed Monte Carlos at low simulation runs, but only with a minimal margin as the simulation runs increased. The empirical validation and comparison with previous literature support the accuracy of the result, which backs the result in refining previous theories.

For future work, it would be interesting to see when Monte Carlos reaches the same mean deviation values as the genetic algorithm, which would require additional simulation runs. In addition, inserting more faults and conditions could change the outcome of this study since according to Betts and Petty (2016) as the genetic algorithm might not accomplish the same balance between population size and the number of generations. Therefore it could be the case that Monte Carlos might outperform the genetic algorithm for low simulation runs and achieve the same result pattern as Betts and Petty (2016).

In addition, in order for the autonomous vehicle model to handle the inserted fault, and condition, and not deviate from the path, an autonomous steering control is required. It would have been appealing to implement two autonomous steering controls, i.e. active disturbance rejection control (ADRC) and proportional integral derivative (PID) steering control, and compare their robustness by applying the fault and condition (Nowak, Czeczot and Klopot, 2018).

Bibliography

[1] AstaZero, (2018). The Test Site. [online] Available at: http://www.astazero.com/the-test-site/about [Accessed 2018-10-14].

[2] Berger, C. (2014). From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation. journal of Software Engineering for Robotics, [online] Volume 5(1), pp. 63-79. Available at: https://arxiv.org/abs/1406.7768 [Accessed 2018-10-16].

[3] Baekgyu, K., Yusuke, K., Siyuan, K., Shinichi, S. (2016). Testing Autonomous Vehicle Software in the Virtual Prototyping Environment. journal of IEEE Embedded Systems Letters, [online] Volume 9(1), pp. 5-8. Available at: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/document/7797233/ [Accessed 2018-10-16].

[4] Rauskolb, F.W., et al. (2008). Caroline: An Autonomously Driving Vehicle for Urban Environments. journal of Field Robotics, [online] Volume 25(9), pp. 674–724. Available at: http://dx.doi.org/10.1002/rob.20254 [Accessed 2018-10-16].

[5] Betts, K., Petty, M. (2016). Automated Search-Based Robustness Testing for Autonomous Vehicle Software. journal of Modelling and Simulation in Engineering, [online] Volume 2016(2016), pp. 1-15. Available at: https://doi.org/10.1155/2016/5309348 [Accessed 2018-10-16].

[6] OpenDaVINCI, (2018). OpenDaVINCI. [online] Available at: http://opendavinci.cse.chalmers.se/www [Accessed 2018-10-14].

[7] Kang, Y., Yin, H., Berger, C., (2018). Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments. Special Issue of IEEE Transactions on Intelligent Vehicles (submitted).

[8] Bogdan, K. (1990). Automated Software Test Data Generation. journal of IEEE Transactions on Software Engineering, [online] Volume 16(8), pp. 870-879. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/57624 [Accessed 2018-10-16].

[9] Masuda, S. (2017). Software Testing Design Techniques Used in Automated Vehicle Simulations. journal of IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), [online] Volume 10, pp. 300-303. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/7899072 [Accessed 2018-10-16].

[10] Ulbrich, S., et al. (2016). Testing and Validating Tactical Lane Change Behavior Planning for Automated Driving. In: D. Watzenig, M. Horn, ed., Automated Driving: Safer and More Efficient Future Driving, 1st ed. Switzerland: Springer, Cham, pp. 451-471. Available at: https://link-springer-com.proxy.lib.chalmers.se/chapter/10.1007/978-3-319-31895-0_19 [Accessed 2018-10-16].

[11] Docker, (2018). Docker Docs. [online] Available at: https://docs.docker.com/get-started/part2/ [Accessed 2018-10-14].

[12] Vohra, D. (2016). Pro Docker. 1st ed. [ebook] CA: Apress, Berkeley, pp. 1-18. Available at: https://link-springer-com.proxy.lib.chalmers.se/book/10.1007%2F978-1-4842-1830-3 [Accessed 2018-10-16].

[13] Thomopoulos, T. (2013). Essentials of Monte Carlo Simulation: Statistical Methods for Building Simulation Models. 2013 ed. [ebook] New York: Springer, pp. 1-55. Available at: https://link-springer-com.proxy.lib.chalmers.se/book/10.1007%2F978-1-4614-6022-0 [Accessed 2018-10-14].

[14] Harman, M., McMinn, P. (2010). A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search. journal of IEEE Transactions on Software Engineering, [online] Volume 36(2), pp. 226-247. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/5342440/ [Accessed 2018-10-14].

[15] Blender, (2018). Blender. [online] Available at: https://www.blender.org/ [Accessed 2018-10-14].

[16] Chu, Z., Sun, Y., Wu, C., Sepheri, N. (2018). Active disturbance rejection control applied to automated steering for lane keeping in autonomous vehicles. journal of Control Engineering Practice, [online] Volume 74, pp. 13-21. Available at: https://www-sciencedirect-com.proxy.lib.chalmers.se/science/article/pii/S0967066118300121 [Accessed 2018-10-14].

[17] Johnson, F.C. (1996). Deductive versus inductive reasoning: A closer look at economics. Social Science Journal, [online] Volume 33(3), pp. 287. Available at: http://eds.b.ebscohost.com.proxy.lib.chalmers.se/eds/detail/detail?vid=4&sid=24cc080a-e0bf-476e-9f9d-48645a0dbe92%40sessionmgr101&bdata=Jmxhbmc9c3

Ymc2l0ZT1lZHMtbGl2ZSZzY29wZT1zaXRl#AN=9607313912&db=buh [Accessed 2018-10-14].

[18] Oyegoke, A. (2011). The constructive research approach in project management research. International Journal of Managing Projects in Business, [online] Volume 4(4), pp. 573-595. Available at: https://www-emeraldinsight-com.proxy.lib.chalmers.se/doi/pdfplus/10.1108/17538371111164029 [Accessed 2018-10-17].

[19] Piirainen, K., Gonzalez A. (2013). Seeking Constructive Synergy: Design Science and the Constructive Research Approach. In: J. vom Brocke, R. Hekkala, S. Ram, M. Rossi, ed., International Conference on Design Science Research in Information Systems and Technology, 8th ed. Berlin: Springer, pp. 59-72. Available at: https://link-springer-com.proxy.lib.chalmers.se/chapter/10.1007%2F978-3-642-38827-9_5 [Accessed 2018-10-17].

[20] Berger, C., Dahlgren, E., Grunden, J., Gunnarsson, D., Holtryd, N., Khazal, A., Mustafa, M., Papatriantafilou, M., Schiller, E. M., Steup, C., Swantesson, V., Tsigas, P., Elad, M. (2013). Bridging Physical and Digital Traffic System Simulations with the Gulliver Test-bed. In Proceedings of International Workshop on Communication Technologies for Vehicles. 5th ed. [ebook] Berlin: Springer, pp. 169-184. Available at: https://link.springer.com/chapter/10.1007%2F978-3-642-37974-1_14 [Accessed 2018-10-17].

[21] Kuhn, D.R., Wallace, D.R., Gallo, A.M. (2004). Software fault interactions and implications for software testing. journal of IEEE Transactions on Software Engineering, [online] volume 30 (6), pp. 418–421. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/1321063/authors#authors [Accessed 2018-10-17].

[22] Zofka, M.R, Klemm, S., Kuhnt, F., Schamm, T., Zöllner, Marius, J. (2016). Testing and Validating High Level Components for Automated Driving: Simulation Framework for Traffic Scenarios. In: Conference for IEEE Intelligent Vehicles Symposium (IV). [online] Gothenburg: IEEE, pp. 144-150. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/7535378 [Accessed 2018-10-17].

[23] Gelperin, D., et al. (1993). An American National Standard IEEE Standard for Software Unit Testing. ANSI/IEEE Std 1008-1987, American National Standard.

[24] Ryser, J., Glinz, M. (1999). A scenario-based approach to validating and testing software systems using statecharts. 1st ed. [pdf] Paris: CNAM. Available at: http://citeseerx.ist.psu.edu.proxy.lib.chalmers.se/viewdoc/download?doi=10.1.

1.22.10&rep=rep1&type=pdf [Accessed 2018-10-17].

[25] Schultz, A.C., Grefenstelte J.J., De Long, K.A. (1993). Test and evaluation by genetic algorithms. 1st ed. [pdf] Washington DC: IEEE. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/stamp/stamp.jsp?tp=&arnumber=236476&tag=1 [Accessed 2018-10-17].

[26] Filho, C.M., Wolf, D.F., Grassi Jr, V., Osório, F.S. (2014). Longitudinal and Lateral Control for Autonomous Ground Vehicles. In: Conference of IEEE Intelligent Vehicles Symposium (IV). [online] Dearborn: IEEE, pp. 588-593. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/6856431 [Accessed 2018-10-17].

[27] Nowak, P., Czeczot, J., Klopot, T. (2018). Robust tuning of a first order reduced Active Disturbance Rejection Controller. journal of Control Engineering Practice, [online] volume 74, pp. 44–57. Available at: https://www-sciencedirect-com.proxy.lib.chalmers.se/science/article/pii/S096706611830011X [Accessed 2018-10-17].

[28] Marino, R., Scalzi, S., Netto, M. (2011). Nested PID steering control for lane keeping in autonomous vehicles. journal of Control Engineering Practice, [online] Volume 19(12), pp. 1459–1467. Available at: https://www-sciencedirect-com.proxy.lib.chalmers.se/science/article/pii/S0967066111001808 [Accessed 2018-10-17].

[29] Lei, B., et al. (2010). Robustness testing for software components. journal of Science of Computer Programming, [online] Volume 75(10), pp. 879–897. Available at: https://www.sciencedirect.com/science/article/pii/S0167642310000328?via%3Dihub [Accessed 2018-10-17].

[30] National Highway Traffic Safety Administration (2008). National motor vehicle crash causation survey: report to Congress. U.S. Department of Transportation, Report DOT HS 811 059. Sringfield: U.S. Department of Transportation, National Highway Traffic Safety Administration. Available at: http://www-nrd.nhtsa.dot.gov/Pubs/811059.PDF (2008). [Accessed 2018-10-17].

[31] Greenblatt, J.B., Shaheen S. (2015). Automated Vehicles, On-Demand Mobility, and Environmental Impacts. journal of Current Sustainable/Renewable Energy Reports, [online] Volume 2(3), pp. 74–81, Available at: https://link.springer.com/article/10.1007/s40518-015-0038-5 [Accessed 2018-10-17].

[32] Gorelov V.A, Komissarov A.I, Miroshnichenko A.V (2015).8×8 wheeled vehicle modeling in a multibody dynamics simulation software. ScienceDirect. Available at: https://doi.org/10.1016/j.proeng.2015.12.066 [Accessed 2018-10-17].

[33] Geyer, S., et al. (2013). Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance. journal of IET Intelligent Transport Systems, [online] Volume 8(3), pp. 183-189, Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/stamp/stamp.jsp?tp=&arnumber=6818481 [Accessed 2018-10-31].

[34] Alsmadi, I. (2012). Advanced Automated Software Testing - Frameworks for Refined Practice. 1st ed. [ebook] Pennsylvania: IGI Global, pp. 1-35, Available at: https://app.knovel.com/web/toc.v/cid:kpAASTFRPF/viewerType:toc/root_slug:advanced-automated-software?kpromoter=federation [Accessed 2018-10-31].

[35] Hooda, I., Chhillar, R. (2014). A Review: Study of Test Case Generation Techniques. journal of International Journal of Computer Applications (0975 – 8887), [online] Volume 107(16), pp. 33-37, Available at: https://pdfs.semanticscholar.org/0de3/f811031c5bafb3ef8ebde008b5485bc1483c.pdf [Accessed 2018-10-31].

[36] Michael, C.C., McGraw, G.E., Schatz, M.A., Walton, C.C. (2002). Genetic algorithms for dynamic test data generation. In: Proceedings 12th IEEE International Conference Automated Software Engineering. [online] Nevada: IEEE, pp. 307-308. Available at: https://ieeexplore-ieee-org.proxy.lib.chalmers.se/document/632858 [Accessed 2018-10-31].

A

Appendix 1

A.1 Test Result

A.1.1 Monte Carlo

A.1.1.1 10 Runs

Run	Mean	Median
1	58.0226091	44.47534
2	61.082397	48.110832
3	41.4932691	41.91173
4	74.5112321	60.475948
5	51.0947034	42.6640775
6	70.8095857	49.284384
7	45.5312985	46.8890175
8	40.7832357	38.8515665
9	43.1313094	41.3324735
10	57.0566532	46.912733

A.1.1.2 20 Runs

Run	Mean	Median
1	43.5203432333333	49.5179655
2	25.7876698166667	20.962808
3	14.23578395	18.19182635
4	29.5170886833333	20.01444005
5	150.540016583333	46.7700865
6	144.093572266667	31.35663975
7	49.3778296666667	22.672227
8	19.5497299833333	7.150823
9	31.4295335	22.260664
10	28.2100230833333	27.75373775
11	71.8236480333333	73.79472825
12	55.2798664666667	43.18205675
13	148.26559495	30.9203945
14	21.4598335333333	20.752234
15	39.5692746666667	25.1128295
16	21.0254594166667	20.6624135
17	37.5470601333333	38.4203774
18	28.8304721833333	30.49612775
19	31.4313027333333	30.43028
20	41.3181262333333	42.90415125

A.1.1.3 40 Runs

Run	Mean	Median
1	66.6060225	44.8511295
2	76.443779	53.11635
3	140.9876029	56.6306955
4	45.5510333	41.13265
5	52.774572	50.92618
6	209.7022653	96.527977
7	125.1961224	46.659851
8	243.8205665	48.781839
9	62.4712829	42.8204725
10	53.0395464	37.8097975
11	90.990112	61.5684335
12	25.2662451	16.0306215
13	26.9451071	19.289444
14	36.2644905	22.8832425
15	13.2872924	8.25349
16	33.7273069	30.3963345
17	33.1508661	18.2528835
18	21.9012548	12.153271
19	32.1986956	21.627301
20	34.1848179	17.7730045
21	42.0942372	31.0549105
22	125.4993608	15.2865565
23	156.4447232	25.5421935
24	168.5059987	36.907383
25	161.6052043	25.203259
26	128.413903	11.3696445
27	252.8820399	60.328518
28	247.7636152	57.256504
29	240.4706618	65.370596
30	109.9099154	31.3342045
31	26.1202075	15.692626
32	33.6960722	29.2253075
33	51.8571978	30.882536
34	24.0796932	21.959922
35	51.2584197	26.936957
36	19.4323081	15.492787
37	38.1579385	23.254797
38	38.0785925	24.361413
39	26.4380024	25.3574395
40	33.2731762	20.00051

A.1.1.4 80 Runs

Run	Mean	Median
1	42.5994465	26.478114
2	36.94751	29.72636
3	52.4189013	40.887188
4	14.5387189	6.1457625
5	527.8404519	664.6416325
6	39.8427865	33.1896965
7	592.9423923	749.317688
8	19.2044957	19.675896
9	32.6952692	25.530388
10	492.9319131	680.707062
11	16.2923998	11.153255
12	56.0993244	29.1252765
13	423.8143267	402.8185505
14	21.3532278	20.903954
15	27.9101129	25.1746225
16	37.3370149	17.529628
17	636.6740357	746.130524
18	34.0987363	32.374035
19	58.0275757	44.611366
20	509.1159903	701.3225095
21	42.4393158	32.1700805
22	35.7771914	32.8612385
23	48.42474	32.025419
24	36.166887	32.6475095
25	48.9765348	43.650383
26	40.8054323	45.03833
27	46.3775557	35.3424785
28	569.4748315	662.3439025
29	467.5615685	626.8547365
30	14.9368749	7.8989095
31	23.8523816	21.216772
32	32.2378338	29.7280045
33	27.1572596	6.550151
34	17.7485566	7.8856585
35	24.6048777	11.5961895
36	43.622318	29.286216
37	584.2273098	692.0310365
38	518.8587837	692.2888185
39	31.1112291	26.3612365
40	33.1106624	28.0944565
41	17.2561568	11.7543955
42	19.5290337	16.0970465

43	15.2914432	8.986588
44	63.2272502	63.117361
45	32.3513983	28.05686
46	19.3731092	11.9154765
47	54.7112129	40.830561
48	18.7249737	15.7059155
49	32.5143857	20.823162
50	52.8774086	40.3993145
51	43.1288951	32.1626655
52	20.9149051	15.5298015
53	25.6112029	16.7256225
54	41.9235002	29.63715
55	40.5169448	39.3768865
56	32.3293845	18.544327
57	668.3244766	747.299469
58	26.9340771	16.517742
59	513.5197541	691.881226
60	342.0518353	208.0229415
61	63.217532	53.4797555
62	44.2318114	32.829649
63	390.4143489	435.3831175
64	373.4045098	371.979519
65	25.5982575	9.578411
66	51.1301914	37.8075895
67	355.7078309	123.4566805
68	419.4674415	361.9779815
69	524.8122817	540.9917295
70	26.2046995	24.1357375
71	25.3422996	23.7855955
72	26.4745683	20.337191
73	473.3328605	660.6095885
74	522.0173202	664.925659
75	36.7848402	35.4412975
76	39.4545181	39.879526
77	419.6932488	399.3829345
78	543.313666	727.2277525
79	328.0296211	108.6963045
80	391.8764657	425.817612

A.1.2 Genetic Alogrithm

A.1.2.1 10 Runs

Run	Mean	Median
1	68.0658761	48.32218
2	55.913046	39.0330425
3	82.3819941	45.860598
4	143.6909927	141.405479
5	169.1914729	141.175362
6	256.4392425	143.480255
7	401.9557741	444.548279
8	454.1401369	444.9919435
9	322.5917973	443.2946625
10	442.304653	536.270172

A.1.2.2 20 Runs

Run	Mean	Median
1	22.8922639	20.7209355
2	26.9921881	24.623374
3	28.4643006	14.5146475
4	38.4250174	40.1272295
5	98.4655625	26.8325605
6	157.7284958	20.229579
7	396.2612723	600.7884825
8	353.5560742	338.6770555
9	649.9062011	670.174225
10	593.9978485	618.0399475
11	563.6591025	745.8158875
12	459.8855639	746.7099305
13	428.3643831	581.9095765
14	294.5941156	61.13913
15	221.6333982	47.374401
16	325.9220889	48.415796
17	116.58747	44.449852
18	292.6190946	67.647751
19	262.5690397	59.330284
20	231.8319092	70.0404015

A.1.2.3 40 Runs

Run	Mean	Median
1	56.0444274	32.0125025
2	43.954622	37.5807935
3	57.0574235	36.956213
4	37.8604417	34.5973935
5	130.5002414	43.497812
6	160.6423015	52.641283
7	502.3731065	656.1812135
8	608.6171936	656.2240905
9	657.9471254	656.9479065
10	606.8718421	660.3384395
11	488.8866185	664.5542605
12	431.8584073	661.166504
13	502.873307	665.218109
14	555.4479828	669.876953
15	611.7211587	671.2715455
16	670.443927	671.2157595
17	543.9120507	670.845642
18	453.2999146	670.8740235
19	433.545669	670.5256045
20	486.7056589	669.9670715
21	345.0636947	265.8549805
22	569.1343239	670.681091
23	384.7066047	461.672119
24	464.5224565	670.456238
25	671.3377075	670.9274295
26	554.9976537	671.1754455
27	425.3920871	671.288696
28	432.4892616	669.610565
29	526.0101413	670.6199645
30	460.5331616	663.7046205
31	288.9714097	200.781715
32	111.9859548	39.0988045
33	287.3009514	87.486519
34	414.198634	587.8728945
35	371.5226691	376.458069
36	438.0618819	597.687805
37	568.859211	670.741516
38	575.0427007	670.284454
39	471.3287469	671.0086975
40	502.7919945	670.6635745

A.1.2.4 80 Runs

Run	Mean	Median
1	28.8483443	25.132599
2	87.1621682	40.5166
3	86.9370099	35.187252
4	147.6613852	44.0630225
5	214.7851052	80.195019
6	321.2081469	291.1485635
7	437.031625	551.062958
8	460.1206198	653.3650515
9	385.8162593	526.609741
10	345.4175139	259.816334
11	379.4989585	470.2821505
12	488.1614487	539.2185975
13	543.4478661	660.8632205
14	606.9815059	664.330994
15	415.4020161	664.3218995
16	368.8700666	444.210983
17	232.4820101	58.6318435
18	160.1994723	50.9627185
19	156.8398631	30.8156765
20	108.5957718	39.593584
21	357.5280308	397.7076035
22	420.4621027	665.5739745
23	401.0311927	437.0374905
24	357.7708277	294.339035
25	312.0317356	309.969864
26	380.3379689	310.28566
27	490.2494602	669.155487
28	531.3321605	669.132507
29	541.9989211	668.9867855
30	535.7576436	671.761597
31	580.2442933	748.1592405
32	581.3912264	749.6625065
33	642.644411	749.042328
34	486.8874471	747.6768495
35	598.2298226	744.944122
36	599.980741	749.2954405
37	590.2863574	748.0543215
38	598.6446762	747.5362245
39	675.9614289	748.047394
40	291.3506944	63.3912905
41	559.9446693	719.170898
42	357.0178716	345.00235

43	450.1460514	696.2993165
44	528.3267799	744.7567745
45	501.5736107	719.5343325
46	536.6569397	748.087311
47	502.8164861	742.5444335
48	678.2190215	749.0365905
49	453.358918	542.2084045
50	380.9245151	389.036896
51	564.5787831	708.448059
52	564.9609134	745.9680785
53	561.0279697	746.7034605
54	577.7100584	745.9752805
55	614.131192	746.7263795
56	750.5616028	751.1838075
57	668.3244766	747.299469
58	592.9423923	749.317688
59	513.5197541	691.881226
60	342.0518353	208.0229415
61	584.2273098	692.0310365
62	518.8587837	692.2888185
63	390.4143489	435.3831175
64	373.4045098	371.979519
65	492.9319131	680.707062
66	423.8143267	402.8185505
67	355.7078309	123.4566805
68	419.4674415	361.9779815
69	524.8122817	540.9917295
70	527.8404519	664.6416325
71	509.1159903	701.3225095
72	636.6740357	746.130524
73	473.3328605	660.6095885
74	522.0173202	664.925659
75	569.4748315	662.3439025
76	467.5615685	626.8547365
77	419.6932488	399.3829345
78	543.313666	727.2277525
79	328.0296211	108.6963045
80	391.8764657	425.817612