Design and analysis of a learning-based testing system for certification of vehicle systems

ADAM MARKROS
Design and Analysis of a Learning-Based Testing System for Certification of Vehicle Systems

Master Thesis
Royal Institute of Technology
Stockholm, Sweden

Adam Markros
markros@kth.se
Acknowledgements

I dedicate this work to my late grandfather, Rahmat Akbar Shokor, who foresaw my potential and encouraged the pursuit of my engineering career.

I would like to acknowledge my gratitude for the endless support from my beloved family, my parents Fredrik and Nasrin Markros, and my dearest of friends; thanks to you this demanding journey has been cheerful and enjoyable. Special thanks to my uncle, Laith Ahmadi, whose advice and expertise has been invaluable throughout this journey.

Further, I would like to thank my supervisor, Lars Drugge, at KTH Royal Institute of Technology for his support and guidance throughout this project. I would also like to thank Alten Sverige AB for giving me the opportunity to conduct this study; with special mention to Detlef Scholle, Staffan Skogby and Christoffer Henriksson, whose consultations have been highly appreciated.

And finally, but certainly not least, I would like to express my gratitude to my fellow thesis workers who made this time entertaining; it has been a joy to work alongside all of you.
Sammanfattning

Abstract

In this work, a learning-based testing system is designed and evaluated in terms of its performance and feasibility of use in testing of safety-critical vehicle systems; the objective is to reduce testing time and costs. A literature study was conducted on the AMASS project, model-based testing and machine learning; based on which a design of the testing system was developed. The finished testing system uses a genetic algorithm for generating solutions of high fitness, which in this application implies test cases that provoke failures in a target system under test, in order for the developers to detect system defects. The target testing system is a model of Volvo’s Brake-By-Wire ABS module. It was concluded that the testing system is effective in increasing fitness of solutions through iteration; the performance of the machine learning algorithm is dependent on parameters such as the mutation rate and the size of the populations into which solutions are clustered.
# Contents

1 Introduction  .................................................. 1  
   1.1 Purpose and Objectives .................................. 2  
       1.1.1 Purpose ................................................. 2  
       1.1.2 Individual Objective ................................. 2  
       1.1.3 Team Objective ......................................... 2  
   1.2 Method ..................................................... 2  
       1.2.1 Theoretical Study .................................... 2  
       1.2.2 Design and Implementation .......................... 3  
   1.3 Delimitations ............................................... 3  

2 Background .................................................... 4  
   2.1 AMASS-project ............................................ 4  
   2.2 Model-Based Testing ........................................ 6  
       2.2.1 Modeling Languages .................................. 6  
       2.2.2 EAST-ADL .............................................. 8  
       2.2.3 UPPAAL ................................................. 8  
       2.2.4 Farkle .................................................. 9  
       2.2.5 NuSMV .................................................. 11  
       2.2.6 Learning-based testing ................................. 11  
       2.2.7 In-the-loop testing ..................................... 11  
   2.3 Machine Learning ........................................... 13  
       2.3.1 Supervised Learning .................................... 13  
       2.3.2 Feature Selection ...................................... 14  
       2.3.3 Unsupervised Learning ................................. 14  
       2.3.4 Reinforcement Learning ............................... 15  
       2.3.5 Machine Learning Algorithms ......................... 16  
   2.4 State of the Art ............................................ 19  
       2.4.1 Model-based testing in automotive applications .... 19  
       2.4.2 Learning-based testing in automotive applications 20  

3 Research methodology ......................................... 23  

4 Specification of Design ........................................ 24  

5 Design ............................................................ 25  
   5.1 Machine Learning Algorithm ............................... 26  
       5.1.1 Initialisation .......................................... 28  
       5.1.2 Iteration ............................................... 28  
       5.1.3 System Under Test ................................... 29  

II
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>CPS</td>
<td>Cyber-Physical System</td>
</tr>
<tr>
<td>MBT</td>
<td>Model-Based Testing</td>
</tr>
<tr>
<td>LBT</td>
<td>Learning-Based Testing</td>
</tr>
<tr>
<td>ATC</td>
<td>Abstract Test Case</td>
</tr>
<tr>
<td>CTC</td>
<td>Concrete Test Case</td>
</tr>
<tr>
<td>ABS</td>
<td>Anti-Lock Brakes</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This master thesis project was done at Alten Sverige AB in collaboration with KTH Royal Institute of Technology in Stockholm, Sweden. The thesis project was a part of the European wide AMASS project [1] which has the objective of reducing time and costs of verifying and certifying Cyber-Physical Systems (CPS) in a number of applications such as automotive, railway and space etc. [2].

Technology in the automotive domain advances rapidly. Modern digital tools have allowed for the development of more advanced functions, increased vehicle safety and lower emissions. Technologies with greater complexity, such as autonomous and connected vehicles are under development. Consequently, modern vehicles have become sophisticated systems with delicate requirements on functionality, interoperability and reliability [2]. It has therefore become an increasingly time consuming and costly process to verify that the systems function in a manner which satisfies its strict requirements.

Embedded systems have become increasingly sophisticated which allows them to facilitate networked systems through interconnections. This feature has been adopted by the automotive domain in order to lay the basis for vehicle platooning. Platooning implies that vehicles can, without drivers, follow a leading vehicle through a wireless connection. The benefits are a reduction in fuel consumption through improved aerodynamics of the convoy, reduction in congestion due to shorter and consistent distances between vehicles and increased safety through elimination of human reaction time. An interconnection of embedded systems of this sort can be classified as a Cyber-Physical System, due to the strong link between physical and digital elements.

The process of verification and certification of such complex systems which can facilitate vehicle platooning is costly and time consuming. However it is vital since the system is expected to perform according to strict requirements in order to be approved for usage. Therefore, in order to ensure proper function of the safety-critical system for vehicle platooning, thorough testing is necessary. The aforementioned AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) project seeks to automate the process through utilising model-based testing (MBT). In model-based testing, models of states which the system under test (SUT) can adopt are used to create test cases. The states which the SUT is able to adopt are based on its predefined requirements. The generated test cases are executed on the SUT and the outcomes are logged and evaluated through comparison with intended results. If the outcome differs from the intended result, the test has failed and post-processing of the result is performed to locate the fault.

It is desired to combine the model-based testing with a learning-based element in order for the system to learn from previous tests and generate new test cases based on them. The objective is to increase thoroughness of the tests to levels above of human capability. Such a testing system would begin with creating testing models based on formalised requirements on the SUT. Test cases would be generated from the models and converted to a format which the SUT can interpret. The tests would be executed and their results would be logged. A machine
learning algorithm would read the logged results, learn from them and generate new test cases.

Another method of automated testing is that of Continuous Integration (CI). CI uses automated testing in order to evaluate code and its functionality. The CI system builds the code and tests it to verify its functionality based on the code’s requirements. It then returns a log of the results from the test, stating whether the code functions correctly or not. CI can shorten development time through continuous testing and detection of errors.

Since vehicle systems are growing in number and in complexity, utilizing a self-sufficient testing system with the ability to verify correct functionality and for certification becomes increasingly suitable for safety-critical systems.

1.1 Purpose and Objectives

This master thesis project is done alongside five other thesis projects at Alten, each concerning a scaled-down prototype car. Each master thesis project has an individual objective and parts of each project should be devoted to achieve a team goal.

1.1.1 Purpose

Testing of safety-critical vehicle systems is vital to ensure correct function such that no harm is caused to humans or the environment. As the complexity of these systems increase, they become more complicated to test and thorough testing becomes increasingly vital. The purpose of this work is to create and evaluate a learning-based testing system combined with in-the-loop testing of an automotive system. This is to provide insight on the combination of the two techniques in order to increase the quality of testing and to reduce testing time and costs.

1.1.2 Individual Objective

In this master thesis, analyses of machine learning and model-based testing are to be conducted. The analyses will lay a theoretical basis for the integration of a machine learning algorithm in a model-based testing system to create autonomously generated test cases. This is in order to incorporate learning-based testing in the system. In-the-loop testing is to be performed and evaluated in combination with the learning-based testing system.

The model-based testing system is provided by Alten and the system under test is a scaled down ABS braking system mounted in a test rig. Previous master theses have studied this model-based testing system, however with emphasis on other aspects. This work will build upon the previous efforts with the aspiration to advance the project further.

1.1.3 Team Objective

The team’s common goal is to give the prototype car two features; one feature is pedestrian detection and the other feature is overtaking of another vehicle if conditions are favourable. These features will be integrated in the same micro-controller.

1.2 Method

This master thesis project will consist of a theoretical study and a practical part including design and implementation.

1.2.1 Theoretical Study

In the theoretical part, a study of literature, academic articles and theses will be conducted on machine learning and model-based testing. Also manuals for software used for model-based testing will be studied. Methods and methodologies will be analysed and their significance for
this work will be evaluated. The theoretical study will be documented and presented in this work.

1.2.2 Design and Implementation

In the practical part, a machine learning algorithm will be created and integrated in an existing model-based testing system in order to incorporate learning-based testing. The finished testing system will be integrated in a micro-controller mounted in a test rig used for analysing specific safety-critical scenarios in a scaled-down prototype car. The system will be tested and verified in order to ensure correct functionality and the results will be presented in this work.

1.3 Delimitations

The limitations of this work are as follows:

- This project is a master thesis in vehicle engineering, thus the project’s focus will be limited to vehicle systems.
- The project’s time-span is limited to 20 weeks, where each week consists of 40 hours of work. A time-plan was made in order to organise the project. 6 weeks of the project’s time-span will be dedicated to the theoretical part, 4 weeks to design of the system, 4 weeks to implementation of the system and 3 weeks for finishing the work.
- The finished testing system is to be implemented in a micro-controller which is integrated in a test-rig for the ABS brake.
Chapter 2

Background

2.1 AMASS-project

AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) is a European-wide project which is to create a tool platform for assuring functionality and certification of Cyber-Physical Systems (CPS) [1]. The target industries are automotive, aerospace and space, railway and energy. The objective of the project is to reduce the cost of certifying CPS. The categories of assurance which collectively constitute the AMASS project are explained below [3].

Architecture-driven assurance

AMASS strives to develop a modeling language, tools and methods with the objective of assuring a system’s architecture. This is in order to analyse the system’s compliance with standards. Architecture-driven assurance include:

- Modeling of a system’s architecture for assurance;
- Verification and validation-based assurance, and
- Assuring new technologies.

Multi-concern assurance

The complexity of embedded leads to the issue of ensuring their dependability e.g. regarding safety, reliability and security [3, 4]. Multi-concern assurance considers the justification that dependability of a system is assured, with most emphasis on safety and security.

Seamless interoperability

When engineering and assuring CPS’, the tools need to exchange data amongst each other. Thus it is required that the systems are well integrated in order to exchange data easily and to facilitate collaboration between different developers.

Cross- and intra-domain reuse

It is possible to reuse assurance data across different versions of a system, projects and domains. However, when reusing data it is necessary to analyse potential consequences. This assurance category concerns process-, product-, argumentation- and cross-concern reuse. The projected technical impacts from the outcomes of the AMASS project include [3]:

- Assurance practices from different domains will merge, thus adopting similar methods and tools;
- More advanced methods for assuring and certifying CPS’, with model-based and simulation technologies;
• Decreased costs of developing and certifying CPS’ due to reuse, and

• Automation of complex processes.

Eleven case studies lay the basis for the AMASS project’s research, where the case studies are distributed over six different domains. In each domain, one or several case studies are analysed: three regard the automotive, three regard space, two regard railway, one regard avionics, one regard air traffic management and one regard industrial automation. The three case studies regarding the automotive industry are:

• $DC$-drive: This case study regards a dashboard used for hardware in the loop (HIL) testing of the DC-driveline of a model car. The objectives of this case study include establishment of formal requirements, automating inspection through fault injection and automating verification through simulation.

• Advanced driver assistance function with electric vehicle subsystem: This case study studies advanced driving assistant systems which has electric actuators. The studies will include models and tools for analysis and verification.

• Collaborative automated fleet of vehicles: This case study analyses cooperative vehicle functions which facilitate governing of the motion of vehicles without driver interference. Specifically, the studied case is that of vehicle platooning: scaled down model cars communicate via WiFi and drive in a controlled environment. The case study includes contact-based safety engineering, simulations with fault injection and automation of safety analysis using monitors.
2.2 Model-Based Testing

In model-based testing, a system is tested using a model of the system that depicts its behaviour in terms of incitation and reaction [7, 8]. The system under test (SUT) is tested using test cases including system conditions before and after testing.

A test case is deduced from a formal model of the SUT; the two types of test cases are: abstract and real test cases. A test case is abstract when its inputs are variables; conversely, a test case is concrete when its inputs are values. Firstly, abstract test cases (ATC) are generated [9]. Next, in order to execute the test case, the ATC needs to be rewritten by assigning variables with real values; this forms the concrete test case (CTC). The CTC is then executed on the SUT; test results are generated and compared to the anticipated outcome. The entirety of this process can be performed manually or automatically. In fully automated model-based testing, the whole operation is automated, ranging from generation of test cases to analysing test results.

A constraint in model-based testing comes from the deduced test cases, the quality of which are strongly dependent on the information contained in the model they are deduced from. Benefits of model-based testing for software developers include [8]:

- Increased efficiency of testing while preserving test quality;
- The system and its behaviour can be reviewed and modeled early in the development process, thus errors can be identified at an early stage;
- Greater transparency in testing, and
- The possibility of measuring and optimising how much the test covers.

2.2.1 Modeling Languages

Describing software with models has become more common since the Unified Modeling Language was established [10]. Modeling languages are used to declare which points of the tested system is to be observed and regulated, describe its dynamic behaviour, and to declare the test data for the test cases. A modeling language connects the model with the requirements on the tested system, and connects the test cases with the test results.

Unified Modeling Language

Unified Modeling Language (UML) provides a standardised notation for graphical modeling, and is the industry standard modeling language for software engineering [11]. Modeling a software graphically is more precise than using natural languages and less detailed than using code; thus graphical modeling is the favoured method since the models give sufficient information of the system with a low complexity [12].

The classes and objects in the software can be graphically illustrated as in Figure 2.1. The illustrations of the classes contain the necessary information about their role in the software [13]. Classes may be sorted into superclasses and subclasses based on commonalities in traits and responsibilities. From their requirements, classes are designated responsibilities. Collaborations signify the interactions between different classes. The classes and responsibilities are analysed in order to distribute the responsibilities and collaborations.

There exists a variety of methods for designing a software system with models; generally it is necessary to identify classes and objects, functions of the objects, interaction between classes, and the realisation of the classes and objects in the form of code. The objective of UML is to express the design models, based on a chosen method by the developer, in a common and standardised language [13].

The UML models which are used to design the system are generally constituted of nine different diagrams, see Figure 2.2. A use case diagram declares the communication between the user and the designed system; a class diagram declares the system’s classes; an object diagram declares the objects in the system in similar notation to that of the class diagram; an activity
Chapter 2. Background

2.2. Model-Based Testing

Figure 2.1: Illustration of a class in UML. Figure from Hunt, John [13].

The diagram contains information of the system’s behaviour in terms of the order in which actions are performed; a sequence diagram contains information which explains the communication between objects which collaborate; a collaboration diagram declares the communication between objects as well as mention the link between the objects; a statechart declares the transitions of states for the objects in the system; a component diagram declares the structure of the source code, and a deployment diagram declares the composition of designed system’s processes and interconnections.

Figure 2.2: The diagrams which constitute a UML model. Figure from: Hunt, John [13].

SysML

SysML is a graphical modeling language which is used to construct systems and to define components, which can be constructed with a domain-specific language, e.g. UML in a software domain [10]. SysML is regarded as an accent of the modeling language UML, and its purpose is to simplify model-based system engineering through constructing a comprehensive model of a system. It can model many features of a system or component, e.g. its structure/architecture, behaviour, restrictions on performance and requirements. SysML is not limited to one domain only, e.g. software; it can be used to model systems in a variety of domains, both man-made and organic systems.

Similar to the UML, the graphical SysML model is constituted of nine diagrams, see Figure 2.3. A package diagram declares how packages consisting of model elements are arranged; a requirement diagram consists of written requirements; a state machine diagram declares the transitions of states for the elements in the system, given some event; a block definition diagram
declares structural elements, named blocks, in the system and their content; an internal block diagram declares the internal connections among the parts in a block, and a parametric diagram declares restrictions on property values.

Figure 2.3: The diagrams which constitute a SysML model. Figure from: Friedenthal et al. [10].

2.2.2 EAST-ADL

Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) is an architecture design language (ADL) [14]. The purpose of an ADL is used to enhance communication, evaluation and analysis in order to assure that the designed system complies with its requirements. The target systems of EAST-ADL are embedded systems with demanding software. An ADL aids the developer in the assurance of proper functionality of the system through supporting verification and validation, model checking and generating code. The purpose of an ADL also includes comprehensible and easy transmission of information between developers.

EAST-ADL was originally realised with the objective to develop vehicle systems; it gives the developer the ability to [14] declare an automotive product’s software and hardware components, its characteristics, specifications and definitions to facilitate analysing the system. EAST-ADL allows the developer to model the architecture of a vehicle’s digital systems in different levels of detail; a higher level of detail is relevant for software whereas a lower level of detail is relevant for hardware components. There are four levels of detail of an EAST-ADL model:

1. **Vehicle level**: The vehicle’s basic elements are declared, e.g. window wipers.
2. **Analysis level**: The basic elements mentioned at vehicle level are further detailed with software which collaborate with the vehicle.
3. **Design level**: The elements are realised at an analysis level; software is further specified with structures and elements, and the architecture of the system’s physical components is specified.
4. **Implementation level**: This is the most detailed level of an EAST-ADL model where AUTOSAR notation is used. AUTOSAR is a tool used for developing vehicle software.

2.2.3 UPPAAAL

UPPAAL is a system which is used for simulating and verifying real-time systems [15]. These are often complex cyber-physical systems; the behaviour of which have been estimated in the form of timed automata [16]. The UPPAAAL system is comprised of three components:
• A modelling language which is used to illustrate the behaviour of the modeled system. This is in the form of timed automata, which means a system that transitions between states when it has fulfilled determined criteria; the time spent at a state being one determining variable;

• A simulator, which is used to detect errors in the modeled system at early development stages. This is achieved through analysing potential behaviour of the system during the development of its model, and

• A model-checker, the purpose of which is to verify the system model’s complete behaviour following its development. UPPAAL’s simulator and model-checker facilitate automated evaluation of the modeled system’s behaviour; this is achieved through analysing the constraints of the system’s state-space model.

UPPAAL allows the developer to model a system both graphically and textually in order to declare the timed automata system and its networks, see Figure 2.4. UPPAAL’s model-checker provides the ability to analyse the model’s reachability, for it is relevant to verify whether nodes are accessible for different combinations of the variables and clocks. As the model-checker analyses a timed automata model’s features, it generates a diagnosis which declares whether the features were satisfied. As aforementioned, UPPAAL also provides the developer with graphical and interactive simulations. The system’s dynamic behaviour is analysed through simulating individual paths through the nodes. The purpose is to detect errors in the modeled system in a more economic manner than the model-checker; this is since the model-checker makes a more thorough analysis of the system’s entire behaviour.

![Graphical Timed Automata Model in UPPAAL](image)

Figure 2.4: A graphical timed automata model in UPPAAL. Figure from Pettersson, Paul et al. [15].

2.2.4 Farkle

Farkle is a tool developed for testing embedded systems [17] [18], initially for embedded systems based on the real-time operating system Operating System Embedded (OSE). For transmission of signals between processes in OSE, LINX is used, which acts an inter-process communication protocol. LINX facilitates the usage of different processors for tasks while communicating in the same manner as on a single processor. Thus Farkle can operate on a separate host machine while testing an embedded system through communicating with the target system through sending input signals and receiving feedback signals. The tests are therefore conducted in the target system’s platform.

Farkle uses abstract test scripts coded in Python for generating inputs to the system under test and to provide test results, see figure 2.5. Test results are generated through the test scripts which contain information regarding expected results of the tests. The abstract test cases are generated from timed automata models of the tested system. For the signal transmission between
2.2. MODEL-BASED TESTING

CHAPTER 2. BACKGROUND

Figure 2.5: Abstract test script where Farkle was applied for verifying the behaviour of Volvo’s Brake-By-Wire ABS system. Figure from Scholle, Detlef et al. [18].

the host and system under test to function, the signals are defined in Python based on a structure defined in source code containing information of the tested system.

The Farkle tool was further developed in order to add additional functionality [18], namely the ability to generate executable test scripts from the abstract test scripts to be executed on the tested system. For the testing purpose, this is relevant because it provides the ability to inspect the state changes at run time in a timed automata model, thus the ability to compare the actual state changes in the system with expected state changes that are declared in the abstract test script. Thus it is possible to determine where errors occur. This version of Farkle was named Extended Farkle. Extended Farkle generates executable test cases and executes them in seven steps:

1. Farkle obtains and reads an abstract test case;
2. The states, transitions between states and variable values are defined from reading the abstract test case;
3. A test script is generated with the information established from reading the abstract test case; this translates to signals containing variable values;
4. The variable values are communicated to the tested system. Initial values provoke the system to transition between states;
5. Signals containing the transitions and states acquired by the tested system as well as variable values are communicated to the test script;
6. The material contained in the signals, regarding the actual behaviour of the system, is compared to the expected behaviour declared in the abstract test case, and
7. A result, pass or fail, is assigned to the test based on eventual inconsistencies between the expected acquired states and actual acquired states. It is possible to also consider comparing expected variable values with actual variable values when concluding the test result.

<table>
<thead>
<tr>
<th>State: (ABSFL.idle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0 ABSFL.y=0</td>
</tr>
<tr>
<td>Transitions: ABSFL.idle-&gt;ABSFL.Entry { w := 8, wheelABS := 1, v := 12 }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State: (ABSFL.Entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSFL.w=0 ABSFL.wheelABS=1 ABSFL.torqueABS=0 ABSFL.y=12</td>
</tr>
<tr>
<td>Transitions: ABSFL.Entry-&gt;ABSFL.CalcSlipRate { v &gt; 0 }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State: (ABSFL.CalcSlipRate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=0 ABSFL.y=12</td>
</tr>
<tr>
<td>Transitions: ABSFL.CalcSlipRate-&gt;ABSFL.Exit { v&lt;6*(v-w)*k, torqueABS := 0 }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State: (ABSFL.Exit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=0 ABSFL.y=12</td>
</tr>
<tr>
<td>Transitions: ABSFL.Exit-&gt;ABSFL.idle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State: (ABSFL.idle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=0 ABSFL.y=12</td>
</tr>
<tr>
<td>Transitions: ABSFL.idle -&gt; ABSFL.Entry { w := 8, wheelABS := 1, v := 12 }</td>
</tr>
</tbody>
</table>
2.2.5 NuSMV

NuSMV is a software developed for checking symbolic models written in the Symbolic Model Verifier (SMV) language, using binary decision diagrams (BDD) [19]. NuSMV facilitates the addition of extensions by other developers, which makes for a flexible tool for many applications. NuSMV is constituted by several modules which provide the program with its functionalities, including:

- Parsing of a model composed in SMV-format. The syntax of the code is checked and a parse tree is generated. The generated parse tree is used by a model compiler in order to generate a finite state machine model, or an automata model, as illustrated in figure 2.4. This model contains a system’s states, transitions between states and conditions which dictate state changes.

- The ability of checking the models and their reachability.

- The ability to check linear temporal logic models (LTL). LTL models are translated into a format which NuSMV can read.

The NuSMV software is equipped with a graphical user interface which facilitates selection of variables and selection of desired functions to use; the functionalities are constituted by eleven packages which transmits commands and routines which the interactive shell employs.

2.2.6 Learning-based testing

Learning-based testing is a form of automated black-box testing of a system’s behaviour; it utilises machine learning for generation of a model of an unknown tested system [20]. Black-box testing means that the inner workings of the tested system is unknown; therefore it is only possible to test the functionality of the system by observing inputs and corresponding outputs of the system.

In learning-based testing, through defining system constraints, test cases are generated containing inputs, \(i\), to the system [21]. As the test cases are executed on the system, corresponding outputs, \(o\), are generated. The pairs, \((i, o)\), are analysed with a machine learning algorithm in order to gain knowledge about the unknown system’s behaviour. This forms a model of the system and from it, new test cases can be generated in order to further expand the knowledge of the system; thus the accuracy of the model grows with the amount of tests executed on it. The generated model is therefore inherently behaviourally correct. Model-based testing methods can then be used to analyse the behaviour of the tested system using the generated model, where test verdicts can be generated using system requirements as input. An example of such a tool is LBTest, see Figure 2.6.

2.2.7 In-the-loop testing

The use of Advanced Driver Assistance Systems (ADAS) in modern vehicles is increasing [22]; these include active safety systems and autonomous systems. Such systems are safety-critical and thus require thorough testing at different stages in their development in order to ensure that they are fail-safe and reliable. The developer therefore defines requirements on the system. To ensure that the system adheres to its requirements, it is necessary to verify and validate the system throughout the development process. In-the-loop simulations are useful for reducing the development time of increasingly advanced ADAS systems; the three main categories of in-the-loop simulation are [22] [23]:

- **Model-in-the-loop, MIL**: A model of the ADAS controller is created, usually with a modeling tool e.g. *Simulink*. This model simulates the logic of the ADAS controller and
interacts with models of the physical system which the controller would be integrated within to evaluate its behaviour.

- **Software-in-the-loop, SIL**: A software code can be created based on the controller model created for the MIL simulations. SIL simulations are then performed to evaluate the generated controller software, using real-time simulations of the physical system.

- **Hardware-in-the-loop, HIL**: HIL testing considers evaluation of the hardware in real-time simulations of the physical system. Some hardware elements may be either real or simulated, where a simulated element would interact with the physical system’s actuators and sensors. If an element is simulated, it should have the same traits as its corresponding real element in terms of inputs and outputs. Benefits of HIL simulations are:
  
  - The repeatability of tests due to a controlled environment.
  - HIL facilitates fault injection, which is relevant for thorough testing of the controller.
  - Systems can be validated sooner during development since the lack och certain components may be replaced by simulations.


2.3 Machine Learning

Machine learning is a category of artificial intelligence (AI) which has advanced rapidly in the last two decades. It utilises statistics in order to train a system to improve from experience and data [21, 25]. Machine learning is used in a variety of domains [26], especially in data-intensive applications. Examples of machine learning applications include, but are not limited to: in e-mail spam detection which learns to distinguish between spam and non-spam mail, in stock trading where machine learning algorithms are used to analyse a stock’s trend in order to decide whether to buy or sell it, and in robotics where a robot can be taught skills through interaction with its environment. Machine learning is divided into three broad categories [27]: supervised learning; unsupervised learning and reinforcement learning.

2.3.1 Supervised Learning

Supervised learning, also known as classification, is the most common category of machine learning [28]. The task is to map previously unseen input data, \( x \), to a class, \( y \), based on previous training [29]. This is illustrated in Figure 2.7 where two classes ‘+’ and ‘−’ have been distinguished by four different classifiers. When the classifier is presented with new, unseen input data, it shall determine which class the data belongs to. In order for the classifier to distinguish between classes, it is trained using pre-classified, labeled, training data i.e. data to which the answers are known. The training is of the form \((x, y)\), where \(x\) is the feature values of an object and \(y\) is the class it belongs to [30].

![Figure 2.7: Four different classifiers yield similar predictions. ‘+’ and ‘−’ represent examples from two different classes, e.g. ‘healthy heart’ and ‘unhealthy heart’. Figure from Domingos, Pedro [28].](image)

Assume that in some domain, it is desired to create a classifier which can classify a sample of data based on certain measurements. In a domain there is a set of objects [31], e.g. patients in the medical domain. A subset of the objects is selected randomly and specific features are measured. The features are correlated to the class, resulting in a vector of measurements for each object. The classifier is then trained using the vectors containing the measured features, the training set.

An example of supervised machine learning in the medical domain is described in the work by Wang et al. [32], where a prediction model was created to predict significant coronary artery disease (sCAD) at an early stage using common laboratory results as input. The selected objects were 1957 patients, of which 1442 had sCAD and 515 did not. Four classification algorithms were tested using laboratory results gathered from the patients: k-Nearest Neighbour, Decision Tree, Random Forest and Support Vector Machine. The measured features consisted of 87 different laboratory markers, of which six different combinations of laboratory markers were considered.
optimal, resulting in prediction accuracies ranging between 77.47 % to 85.63 %. The finished prediction model was concluded to have an accuracy of 85.63 % in predicting sCAD in patients, using Support Vector Machine.

2.3.2 Feature Selection

A feature, in machine learning applications, is an attribute of an object that can be used to describe it [30]. Feature selection is the part of machine learning that requires the greatest effort, one reason being that features are domain-specific, thus domain knowledge is essential [26]. For data analytics, informative features are useful since they portray the observed object and can distinguish between collections of objects. There are several categories of features [30]:

- **Categorical feature**: Discrete values e.g. colour where the domain of features is a range of different colours;
- **Binary feature**: The domain of features has two values e.g. positive and negative;
- **Ordinal feature**: The domain consists of ordered ranks, e.g. Private < Sergeant < Lieutenant, and
- **Numerical feature**: The domain is consists of continuous numerical values e.g. [1, 10].

Objects oftentimes have a large set of features; however by taking only a small set of the features into consideration makes the algorithm more computationally feasible and improves the quality of its result [30]. Removing irrelevant and redundant features also reduces over-fitting, enhances the accuracy of the classifier, decreases complexity and computational time [33]. One method for selecting features is through analysing correlations; namely the correlation between features and the correlation between features and classes. The former is termed feature redundancy and the latter is known as feature relevance. It is desired to select strongly relevant and non-redundant features to obtain an optimal set of features [30].

2.3.3 Unsupervised Learning

Section 2.3.1 depicts a category of machine learning where all training data has a class label. However, in many applications the data is not labeled; this is the type of data that is treated in unsupervised training. In some applications however, only parts of the input data is labeled; this category of machine learning is a combination of supervised and unsupervised learning and is named semi-supervised learning. In unsupervised learning, data is processed in order to highlight characteristics in the data, in order to find patterns [26]. There are two main characteristics in the data that are analysed [34]: the data’s subspace structure which is explored through feature reduction and the data’s clustering characteristics which is explored through clustering by using less objects than the original measurement.

The method of clustering has the objective to distinguish between objects in the data through dividing the data into subsets, where the objects in each of the subsets have greater resemblance with each other than with objects in other subsets. The resemblance between objects is generally calculated as the distance between them; mathematical measures such as the Euclidean distance and the Cosine angle are commonly used to measure resemblance. The generated clusters are named in order to be used for making predictions on future data [24, 25].

Genetic algorithms is a category of machine learning which applies the theory of evolution for solving optimisation problems [35]. A genetic algorithm takes a population of conceivable solutions to the optimisation problem and generates an optimal solution based on them through a series of iterations. The iterations mimic those of a Darwinian evolutionary process, as the fittest solutions are favoured; the reward of a greater fitness is a higher likelihood of procreation. The ability to solve the optimisation problem for each of the conceivable solutions in the population need to be assessed. For this a fitness function is needed, which is a function
that quantifies the quality of a solution in terms of how well it solves the problem. The fitness function assigns a fitness score to each solution, which is vital for the selection process. In order for the genetic algorithm to converge to an optimal solution, it aspires to select solutions with higher fitness for procreation. There are a variety of selection processes, of which elitist selection is most common, meaning a strict bias towards fitter solutions. The pie chart algorithm is an effective elitist selection algorithm. An illustration is given in Figure 2.8 which displays a pie chart representing the fitnesses of four different solutions. All solutions are represented and their likelihood to be selected for procreation is proportional to their fitness score.

![Pie chart illustrating fitnesses of four solutions](image)

Figure 2.8: A pie chart where the fitnesses of four different solutions are illustrated.

In a population of $N$ solutions each with a fitness score of $w_i$, the probability of solution $i$ to be selected for parenting offspring is

$$p_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

(2.1)

After the selection of parent solutions, a set of new solutions is generated from them. Two or more solutions are required as parents for procreation, since the procreation involves crossover of feature values between the parents. The motive is that combining features of solutions that score high in fitness is likely to produce offspring that outperform their parents. Bit strings can provide a simple example of a crossover: The two parents are 1001110101 and 0011001011; the crossover process is to split the parent bit strings in two halves and combine the halves in different sequences to generate a new population, e.g. 10011-00110, 10011-01011, 10101-00110 etc.

A complement to crossover is mutation, which makes random alterations to the offspring. Mutations occur with a mutation rate, which is defined by the developer, and the purpose is to increase the reachability of the offspring solutions. Mutations which result in undesirable outcomes will not survive due to low fitness; the vice versa condition also holds. Depending on the application, the genetic algorithm can be terminated after a predefined number of iterations or when sufficient convergence to an optimal solution is achieved.

### 2.3.4 Reinforcement Learning

Reinforcement learning differs from supervised and unsupervised learning in the sense that its purpose is not to indicate the correct class given input data. Rather, a reinforcement learning algorithm is trained on data which indicates whether actions are acceptable or not. In the case of an unacceptable action, the issue becomes deciding an acceptable action. The data is generated by a reward system, where each action produces a certain reward. Based on the reward system, the reinforcement learning algorithm learns a strategy to select actions
which maximises the reward. Creating a reward function is commonly easier than mapping observations, from e.g. sensors, into actions [37].

Reinforcement learning is useful for e.g. teaching robots how to perform tasks by specifying the tasks and letting the robot learn the details of how to perform the tasks through trial and error. The work by Kaelbling et al. [37] provides a valuable example of such an application, namely a mobile robot which was to reach a target point while avoiding obstacles. The authors formed a reward function, \( R(s, a) \), which takes the state, \( s \), and action, \( a \), into account and returns a reward. The algorithm was trained in two steps; the first was a passive phase where the robot was controlled by either a human via a joystick or via a control algorithm, and observing the states. In the second training phase, the algorithm controlled the robot based on its training from the first phase; since the robot could find reward giving states due to the first training phase. After 35 iterations of second phase training, the algorithm’s performance was equivalent to that of the case when the authors controlled the robot manually in when driving in corridors. Likewise in obstacle avoidance, the algorithm’s performance was equivalent to that of the authors’ when fully trained. The reward function was simple; the algorithm earned a reward of ’1’ when reaching the destination, and a reward of ’-1’ was earned when coming in contact with an obstacle; a reward of ’0’ was given in any other situation. The authors concluded that the algorithm could learn to complete the desired tasks in less time than what it would take programmers to hand-code the robot to complete the tasks. It was also concluded that when guiding the robot, the human did not need to demonstrate the best solution to the robot; rather it uses the examples to generate experience to improve upon.

![Figure 2.9: Success in avoiding collision with obstacles; each point on the horizontal axis corresponds to ten runs. Figure from: Kaelbling, Leslie et al. [37].](image)

### 2.3.5 Machine Learning Algorithms

Different machine learning algorithms can be used depending on the category, i.e. supervised, unsupervised and semi-supervised applications [38]. In classification, the input data is a set of features which a classifier aims to classify based on the data’s traits. In order for the algorithm to match the input data with a class; a classification algorithm needs to be trained. Training a classification algorithm means adapting its parameters for it to couple the data to a class. There exists a variety of classification algorithms, e.g:

- **Decision trees**: This is a simple classification method which requires little data preparation, is easy to comprehend and to validate [26]. A decision tree consists of nodes; each node is related to a feature of the data. The nodes are connected by branches which represent the possible outcomes of the node. Data is classified through following the tree structure, from the root to the leaf, see Figure [2.10]. The leaves represents the classes.
2.3. MACHINE LEARNING

Figure 2.10: A decision tree classifier which predicts whether a person is allowed a loan or not. 'Age' is the root node. Two outcomes of the age are possible: older or younger than 55 years. These branches lead to different nodes: 'home owner' or 'good credit'; of which two outcomes, 'Yes' and 'No', are possible. These branches lead to the leaves, i.e. the class, which represents whether the person is allowed a loan or not. Figure from Bell, Jason [26].

- **Support Vector Machine**: Support vector machine (SVM) determines hyper planes which divide data points of the respective classes, see Figure 2.11 and gutters which give margins between the classes and the hyper planes. Training the SVM is done through determining the equations for the hyper planes, i.e. $\vec{w}$ and $b$ as illustrated in Figure 2.11. SVM then classifies new data points, $\vec{x}$, by determining on which side of the hyper plane the data point is located.

Figure 2.11: The line $\vec{w} \cdot \vec{x} + b = 0$ represents the hyper plane dividing the two classes, illustrated as white and grey data points. The lines, $\vec{w} \cdot \vec{x} + b = \pm 1$, represent the gutters.

Clustering algorithms, which is a part of unsupervised learning, strive to categorise data points. Common clustering algorithms are [39]:

- **First K**: A high speed clustering algorithm which finds cluster centers. It indicates the first $K$ data points which it finds as the centers, and assigns following data points to the
closest cluster center. The distance between a data point and a cluster center is calculated as the Euclidean distance;

- **K-means**: A common clustering algorithm which requests the amount of clusters the data contains as an input, after which K data points are randomly selected in order to shape the early cluster centers. As in *First K*, following data points are assigned to the respective clusters by calculating the distance to the centers with the Euclidean distance;

- **Farthest first**: Data points are selected randomly from the data set to create cluster centers. The algorithm then creates the following cluster centers as the data points with the furthest distances to the existing centers. The distances are calculated with the Euclidean distance;

- **Bisecting K-Means**: Based on the K-means algorithm, Bisecting K-means sets $K = 2$ which generates two clusters. The largest of the bisected clusters is then bisected into two clusters and this is repeated until the amount of clusters is equivalent to that which is inputted by the user.
2.4 State of the Art

2.4.1 Model-based testing in automotive applications

The work by Scholle et al. \cite{18} describes a technique for verifying the behaviour of the automotive manufacturer Volvo’s Brake-By-Wire (BBW) braking system, in which the conventional mechanical and hydraulic components are replaced by electronic actuators and sensors. The BBW braking system needs to retain the anti-lock brakes (ABS) function in order to avoid the wheels locking during braking; each wheel’s slip rate is calculated and based on this, the corresponding brake actuator engages by applying a desired braking torque or disengages if the wheel’s slip rate has exceeded a predefined limit. Therefore, the BBW is a safety critical system and it is essential that its function conforms with its specifications.

Extended Farkle was used to verify the BBW system’s behaviour, see section 2.2.4. Therefore the testing method requires the generation of concrete test cases from abstract test cases. The BBW system’s architecture and components were modeled in EAST-ADL, see section 2.2.2. Timed automata models of the components comprising the BBW system were then created using UPPAAL PORT, which is an extension of UPPAAL, see section 2.2.3. Figure 2.12 presents a state machine model of an ABS module used in the study \cite{18}.

![Figure 2.12: ABS module of the BBW system expressed as a timed automata model. Figure from: Scholle, Detlef et al. \cite{18}.](image)

Through declaring timed computational tree logic criteria in the UPPAAL PORT model, abstract test cases could be generated. Concrete, executable test cases were generated as described in section 2.2.4 from the abstract test cases and executed on the BBW system through signals from a host machine. The system’s state changes and variable values were logged and communicated back to the host machine; the logs of the system’s states and variable values were compared with the expected behaviour declared in the test scripts. The method also has the ability to locate where an eventual error has occurred, which facilitates troubleshooting for the developer. If an error is located, the authors of \cite{18} suggested generating more test cases that aim to test the related states and paths of the system, which can aid in the discovery of the error’s origin.

In summary, the method described in this work allows for testing of a system’s states and transition between states for verifying its behaviour. The testing was conducted through comparing the system’s run-time behaviour with its desired behaviour expressed with models. It needs to be acknowledged that a successful test case does not equate an defect free system; rather it signifies an increased certainty of the system’s quality.

It is desired to further develop the method such that it is able to analyse more than one component, thus analysing the link between components. The purpose is to test patterns in communication between the components as well as analysing changes in the architecture e.g. through substituting components. Further, the authors suggest the incorporation of a feedback loop which connects the models and the test results. This is of interest because when an error is identified at run-time, the corresponding path can be observed further either with a static analysis in order to study the error, or with a dynamic analysis through automatically creating...
more abstract and concrete test cases that target the error prone path which leads to further
testing of it.

2.4.2 Learning-based testing in automotive applications

The work by Meinke et al. [20] discusses learning-based testing for verifying the behaviour of
vehicle systems. Two cases were studied in association with the truck manufacturer Scania,
both regarding vehicle ECU’s. As described in section 2.2.6 learning-based testing generates
behavioural models of a system through machine learning; the generated models can then be
used for testing using model-based testing methods. This approach is advantageous in cases
when there is no knowledge about the inner workings of the tested system, i.e. the system is
a black box; thus through the use of machine learning it is possible to generate models despite
the lack of knowledge about the system.

As aforementioned, the authors studied two systems provided by Scania:

1. Engine Start Application (ESTA), a remote engine starting system. The authors focused
on modeling the system’s informal behavioural requirements and to find whether it is
possible to formulate informal behavioural requirements in linear temporal logic (LTL).

2. A dual circuit steering system (DCS). The authors focused on formalising informal require-
ments in LTL, detecting unknown errors in the DCS-system and to compare the LBTest
tool with another testing tool.

ESTA-system

ESTA engages when a signal is received; an analysis of the vehicle conditions is conducted and
the engine is started if the conditions are favourable, otherwise it is left shut off; Figure 2.13
presents the schematics of the ESTA-system. The system is implemented within an ECU and
communicates through CAN-signals; it receives inputs from switches and sensors and outputs
signals to the engine and the vehicle’s indicators.

![Figure 2.13: A schematic illustration of the ESTA-system. Figure from: Meinke, Karl et al. [20].](image)

System requirements were established through deciding input and output parameters, which
were either numerical values within determined ranges or Boolean (true/false). Due to a vast
set of input parameters, the size of the input set was scaled down by combining them pairwise,
since the input parameters belonged to one of two classes.

Behavioural requirements were declared in natural language; these were translated into PLTL
code, which is a language supported by LBTest to create test cases. Using an available timed
automata model from Scania, states and transitions were also translated to PLTL code. The
timed automata model acted as a reference model for ESTA, therefore requirements could be generated as in conventional model-based testing. Based on the PLTL requirements used by LBTest, learning of a model was conducted to potentially discover disagreements between the model’s behaviour and the system’s requirements. Due to a vast set of input parameters, the convergence was low. Thus the size of set had to be decreased through exclusion of some parameters; thereafter the convergence increased to a decent degree.

**DCS-system**

The purpose of a dual circuit steering system is to retain steering-ability in case of a failing hydraulic system. The DCS-system is a software implemented in an ECU belonging to the vehicle’s secondary steering circuit. Figure 2.14 presents the schematics of the DCS-system. Similar to the ESTA-system, the DCS-system receives signals via CAN-signals in order to evaluate the state of the vehicle. The system’s input values are either discrete or continuous whereas the system’s output parameter values are discrete only.

![DCS-system diagram](image)

**Figure 2.14: A schematic illustration of the DCS-system. Figure from: Meinke, Karl et al. [20].**

Based on declared limits on parameter values found in the system requirements provided by Scania, samples of the input and output parameter domains were gathered. With some difficulty due to the format of the provided system informal requirements, these were successfully converted to PLTL black box requirements. Based on the black box requirements, a learned model was generated, constituting of more than 60 states and an excess of 800 transitions; the model had a convergence of 97 % to the system requirements.

As in the case of the ESTA-system, the tool used to generate and evaluate the learned model was LBTest. However in the DCS case, analyses were conducted of LBTest’s ability to detect unknown inconsistencies between the tested system and its requirements, as well as its ability to detect injected faults:

- Five unknown inconsistencies between the tested system’s behaviour and its requirements were discovered. It was through further investigation concluded that the discovered inconsistencies were actual errors with the system, however not critical.
- LBTest’s ability to detect injected faults was tested; the faults were implemented in the source code. The injected faults were in the form of altered boundary values, disordered input and output parameters as well as altered Boolean statements at random. A total of ten faults were injected, of which LBTest detected eight. For comparison, Scania’s tool piTest detected two of the injected faults.

This study was conducted to analyse the method of learning-based testing for testing requirements on the behaviour of vehicle ECU systems. Regarding the learning-based testing, the authors wanted to study:
• The possibility for a learner algorithm to generate a model which represents the entire behaviour of an ECU-system, on a detailed level within an acceptable time-frame.

• The possibility to examine the generated model and to create test cases which can cause legitimate test failures within an acceptable time-frame.

Based on the results of the learned model of the DCS-system, it is possible to learn a model which represents the entire behaviour of a system on a detailed level. The model can also be checked and it can cause legitimate test failures, in order to study the system’s unknown defects. The authors’ conclusions from the study are that PLTL is a useful method for representation of a system’s behavioural requirements; the authors also concluded that learning-based testing is an appropriate method for black box testing software, one reason being that the models generated by machine learning inherently conform with the tested system’s behaviour, which is not assured in models used in conventional model-based testing. For further research, the authors advocate virtualised hardware-in-the-loop testing, which facilitates fault injection.
Chapter 3

Research methodology

When conducting research, it is essential to support the work with methods and methodologies, as these determine the results of the work [40]. A method is the procedure which is used to formulate facts; a methodology is the rationale behind a method and the justification of using a certain method in favour of others. Methodologies can be divided into two categories: quantitative and qualitative. Methods are then chosen based on the methodology which the project adheres to. Quantitative methodologies require substantial amounts of data for generating conclusions and demand statistics to verify the hypotheses, whereas qualitative methodologies use modest amounts of data and involve analysing behaviours to establish theories.

This work was initiated with a study on the background of the subject. Such a background study is significant for acquiring the information necessary for planning of the project in order to obtain the desired result. The background study serves as a theoretical foundation of the problem topic, based on which a methodology could be selected and a research problem formulated. The methodology of this work is qualitative; conclusions are drawn more from behaviours than from quantifying variable values and utilising statistical methods to confirm hypotheses.

The philosophical assumption which leads this work is realism. This implies that reliable data is gathered from in-the-loop testing in combination with a learning-based testing system; the gathered data is used to further increase the understanding of the concept. Research methods instruct how the research is to be performed, suggesting methods which aid the course of the research. Fundamental research was the method chosen for this work; this implies that a phenomenon is observed in order to further develop the understanding of its behaviour, often used for innovation purposes.

In order to formulate conclusions, the work needs to follow a research approach. There are two main research approaches: inductive and deductive. An inductive research approach implies that conclusions are generated based on behaviours, whereas a deductive research approach implies that hypotheses are tried and it is sought to discover causal relationships using large sets of data. This work follows an inductive research approach, since it is sought to analyse and evaluate behaviours.

An inductive research strategy is that of grounded theory, which establishes theories which can be proven with data. This research strategy strives to increase the knowledge of the features of what is observed. The method for gathering data in this work is through observations, as behaviours are be analysed. For formulating conclusions in research projects, the gathered data needs to be evaluated. A combination of two methods are used for analysis of the data. The first method, computational mathematics, is a quantitative method which is generally used for calculation but also for simulation and modeling. The second method, narrative analysis, is a qualitative method which strives to maintain the traceability of requirements.

For quality assurance, the transferability of this work is assessed; this implies that the work is described comprehensively in order to facilitate further research.
Chapter 4

Specification of Design

Selection of a suitable machine learning algorithm is not a straightforward task, since there exists a wide variety of algorithms with the same objective. For this work, the genetic algorithm as described in Section 2.3.3 was chosen as algorithm for generation of test cases. The justifications for choosing this algorithm are the fact that it can operate without a demand of derivatives and its versatility as it can be adopted by a wide range of domains. Due to the peculiarity of this project, the use of an open-source algorithm was not feasible; therefore an algorithm was created for the project. The programming language, Python, was considered suitable for the task as it supports a wide variety of packages and extensions.

At the beginning of the project, the goal was to integrate the machine learning algorithm with the Farkle testing tool, as described in Section 2.2.4 within AMASS’ DC-drive test rig, as described in Section 2.1. However, due to the inability to reassemble the Farkle testing tool, an alternative solution was needed in order to verify and validate the machine learning algorithm. Therefore a state machine model of an ABS module was created with the purpose of imitating the physical system under test which the Farkle testing tool interacts with. The system will thus perform model-in-the-loop testing rather than hardware-in-the-loop testing as initially intended; see Section 2.2.7 about in-the-loop testing.
Chapter 5

Design

The task of the machine learning algorithm is to generate abstract test cases (ATC) to be executed on the system under test; the algorithm is to learn the circumstances which provoke failures in the system in order for the developer to discover the system’s defects. The system under test is the front left anti-lock brake (ABS) module of Volvo’s Brake-By-Wire (BBW) system, which was studied in [17, 18]. Through a study of the files used in the aforementioned studies, the system in this work could be constructed. Five handwritten ATC’s were provided, which are input files compatible with the Farkle testing tool, see Appendix A.1. The ATC’s differ in the target component tested in the system under test.

All ATC’s follow a specific pattern, namely: state → parameters → transition, see Appendix A.1. For correct parsing of the ATC’s, the pattern must be maintained. State signifies the state of the system’s modules, of which there are two: Global Brake Controller (GBC) and ABS Front Left (ABSFL). Thus, GlobalBrakeController.idle signifies that the GBC module is idling at that state; all modules begin and end the test at idle state. There are 14 parameters with corresponding numerical values in the ATC’s, of which the wheel radius, \( R \), remains unchanged. \( R \) and \( x \) exist in both modules. The parameter values may change between different states; see Table 5.1 for all parameters. Note that parameters \( x \) and \( R \) recur twice in the ATC but are only mentioned once in the table. Transitions signify the transitions between two states and are completed with conditions on the parameters which need to be fulfilled in order for a transition change to occur. Each ATC has a unique shortname for traceability.

Table 5.1: The parameters considered in the BBW system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>Time</td>
</tr>
<tr>
<td>( rpm1 )</td>
<td>Rotational velocity of wheel 1</td>
</tr>
<tr>
<td>( rpm2 )</td>
<td>Rotational velocity of wheel 2</td>
</tr>
<tr>
<td>( rpm3 )</td>
<td>Rotational velocity of wheel 3</td>
</tr>
<tr>
<td>( rpm4 )</td>
<td>Rotational velocity of wheel 4</td>
</tr>
<tr>
<td>reqTorque</td>
<td>Required braking torque</td>
</tr>
<tr>
<td>whlTorque</td>
<td>Actual braking torque</td>
</tr>
<tr>
<td>( v )</td>
<td>Speed of the car</td>
</tr>
<tr>
<td>( R )</td>
<td>Wheel radius</td>
</tr>
<tr>
<td>( w )</td>
<td>Rotational velocity of wheel [rad/s]</td>
</tr>
<tr>
<td>wheelABS</td>
<td>ABS activity</td>
</tr>
<tr>
<td>torqueABS</td>
<td>ABS torque</td>
</tr>
</tbody>
</table>

The machine learning algorithm was intended to work autonomously in conjunction with the Farkle testing tool. In parts of Farkle’s source code the ATC’s are parsed and tables containing information about the states, transitions and parameters are generated. The source code is
written in the coding language *Ruby*, and the machine learning algorithm in Python. Therefore, parts of Farkle’s source code was modified for storing the aforementioned tables as database in comma separated values (CSV) form, see Appendix [A.2]. A script, *readtables.py*, was written in Python for parsing of the tables in the database, accumulating parameter values, states, transitions and an optional test verdict in a struct.

### 5.1 Machine Learning Algorithm

As stated in Chapter 4, the machine learning algorithm chosen for this application is a genetic algorithm. As described in Section 2.3.3 a genetic algorithm requires a population of solutions; an algorithm for calculating the fitness of each solution; an algorithm for selecting parent solutions, and an algorithm for generating offspring through crossover and mutation of selected parents.

In the case where populations would consist of individual test cases, the range of fitnesses to select from would be binary (pass and fail). Thus a large population would consist of multiple solutions with equally high and equally low fitness scores, resulting in difficulties in selecting parents. For selection it is preferred that the range of fitness scores is wide, which results in a better ability to rank the quality of solutions in the population. Therefore the populations were set as groups of solutions, where each solution is an ATC, see Figure 5.1. This gives the ability of ranking on a non-binary scale. Since the purpose of the testing tool is to detect failures in the system under test, the more failures a group of ATC’s generates, the higher the fitness of the group. Fitness score was calculated as the sum of failed ATC’s in a group. An example: Suppose that a population contains 10 groups, each consisting of 5 ATC’s with a unique parameter configuration of numerical values of the parameters mentioned in Table 5.1. For each ATC that generates a failure, the group it belongs to gains a fitness score of 1; thus the maximum fitness score a group can have is equal to the amount of ATC’s it contains, which is 5 in this example. Suppose therefore that the fitness scores of the population are \([3, 1, 1, 0, 2, 0, 4, 1, 0, 1]\), thus group 7 has the highest fitness score where 4 of its 5 ATC’s result in failures.

![Figure 5.1: Illustration of a population. The population consists of groups which contain ATC’s, each with a unique set of parameter values.](image)

A selection algorithm is required for selecting parents for procreation. For the purpose of this work, an elitist selection algorithm as described in Section 2.3.3 is preferred. Elitist selection algorithms are biased towards solutions with higher fitness scores; the pie chart algorithm is such a selection method which gives opportunity for all solutions to parent offspring. However, their probability of being selected is proportional to their fitness score, meaning that a higher fitness score yields a higher likelihood of parenting offspring. According to Equation 2.1, the probability of a group \(i\) with corresponding fitness \(w_i\) in a population of \(N\) groups, to be selected is

\[
p_i = \frac{w_i}{\sum_{i=1}^{N} w_i}
\]
This was expressed in code as presented in Figure 5.2.

```python
def piechartSelection(self):
    rand = random.randint(0, totalsum)
    partialSum = 0
    for each in range(0, len(self.group)):
        partialSum += self.groupFitness[each]
        if partialSum >= rand:
            return each
```

Figure 5.2: Pie chart selection algorithm expressed in Python.

The pie chart selection algorithm selects one group of ATC’s for parenting offspring consisting of a new set of \( N \) groups of ATC’s with unique parameter sets. The higher the fitness score of the parent group, the greater the likelihood is that the offspring will outperform their parents.

The offspring ATC’s are generated through crossover and mutation of the parent ATC’s and clustered into groups. In the crossover function, all values from a randomly selected parameter in a randomly selected ATC is extracted; the extracted parameter values replace the corresponding parameter values in the remaining ATC’s. This procedure is performed until a sufficient amount of ATC’s have been generated for creating a population of \( N \) groups of ATC’s. Mutation is vital in the generation of offspring; it increases the reachability of the offspring since crossover alone has a limited range of conceivable parameter values. Mutation occurs with a mutation rate, a probability \( p_{\text{mutate}} \in [0, 1] \). For each iteration of crossover, the probability of a mutation occurring is \( p_{\text{mutate}} \). The mutation function introduces a random integer in one instance of a randomly selected parameter, see Figure 5.3. As aforementioned, the value of the wheel radius \( R \) remains unchanged; the crossover algorithm therefore neglects this parameter.

![Figure 5.3: Crossover of parameter values between two parent ATC’s. The rows in the matrices correspond to states, the columns correspond to parameters. Note the mutation indicated in the figure.](image)

27
5.1. MACHINE LEARNING ALGORITHM

5.1.1 Initialisation

In order to initialise the process of generating test cases, an initial population of ATC’s is required. A function was written for this purpose as a part of the machine learning code. Initially, the ATC’s are read through calling the aforementioned `readtables.py` script which dissects the ATC’s and returns a struct containing its parameter values, states, transitions and test verdicts. A set of ATC’s, each with a unique parameter configuration, is generated and clustered into $N$ populations containing $n$ ATC’s each. In order to increase the reachability of the test cases, the algorithm for generating an initial population of ATC’s works as follows: The original parameter configuration of an input, handwritten, ATC is parsed from `readtables.py`. Thereafter, each variation of the parameter configuration is generated with one of three means, namely

1. One single value is replaced with a randomly generated integer;
2. One entire parameter is increased with a randomly generated increment, or
3. Parameter values of one entire state is increased with a randomly generated increment.

The ATC’s in the generated initial populations are tested in the tester algorithm which returns a test verdict for each ATC, i.e. whether a test case has passed or failed, see Section 5.1.3. Based on the test verdicts, fitness scores can be calculated and assigned to each population.

5.1.2 Iteration

After the initialisation process described in Section 5.1.1, the genetic algorithm iterates new populations of ATC’s based on previous populations’ fitness scores, with the objective that the offspring will outperform its parents. After the initialisation process, an iterative process takes over, see Figure 5.4. From the new population generated in the initialisation process, a parent population is selected using the pie chart selection algorithm as described above; the offspring ATC’s are generated from the ATC’s in the parent population using crossover and mutation as described; the offspring ATC’s are clustered into populations; the ATC’s in the generated populations are tested on the target system under test described in Section 5.1.3 which generates verdicts for each ATC; failed ATC’s are stored in database; fitness scores are calculated and assigned to the populations, and the ‘loop’ starts over.

The system allows the user to select the amount of iterations of procreation until termination of the algorithm; the mutation rate, which signifies the probability of mutation; the amount of populations; the size of populations, which in this application signifies the amount of ATC’s contained in a population; and the range of integers which the mutation algorithm can select from.
5.1.3 System Under Test

As mentioned in Chapter 4, a state machine model of an ABS module was created for the purpose of testing a target system in order to validate the machine learning algorithm. The ABS module was derived from the module described in the work by Scholle et al. [18] regarding Volvo’s Brake-By-Wire system, see Section 2.4.1. Figure 5.5 shows a graphical illustration of the model created for this work; the module was expressed in Python code. The model calculates the path taken in the state machine model based on the parameter values given in the ATC. A verdict is generated based on the compliance between the ATC and the behaviour of the ABS module; e.g. if \( v = 0 \) but \( \text{wheelABS} = 1 \), the verdict is ‘fail’ since the system applies ABS when it is not supposed to.

\[
\begin{align*}
\text{Entry: } v > 0 \\
\text{CalcSlipRate: } v < 5(v - wR), \text{wheelABS} = 0 \\
\text{Exit: } v \geq 5(v - wR), \text{wheelABS} = 1 \\
\end{align*}
\]

Figure 5.5: Illustration of the ABS state machine model created for this work.
Chapter 6

Results and Verification

The system consisting of the genetic algorithm described in Chapter 5 was evaluated in terms of its ability to increase the fitness of solutions through iteration, the results of which are to answer the initial question of the feasibility of combining in-the-loop testing with learning-based testing. The results are presented graphically and are obtained using different configurations on the mutation rate and population size; the configurations are compared and evaluated. Each figure contains two graphs; the left is a bar diagram which presents the fitness scores of the populations, comparing the fitness scores obtained in the first iteration with those of the last iteration; the right is a graph which presents the total sum of failures among all populations at each iteration, which demonstrates the learning rate of the algorithm as a function of iterations.

6.1 Influence of Mutation Rate

Five populations with a population size of 10 ATC’s each were used while the mutation rate was varied and its effects were analysed. In order to emphasise the effect of the change in mutation rate, identical initial populations were used. Figure 6.1 presents results obtained with a mutation rate of 3 %, a population size of 10 ATC’s and 50 iterations of procreation. From the bar diagram of Figure 6.1, it is apparent that at the end of the iterations, 10 out of 10 ATC’s in each population has a fitness score of 1, meaning that each ATC generates a failure. The graph in Figure 6.1 shows a learning rate which rapidly results in a maximal fitness score of 50, which is the total amount of ATC’s among all populations. Slight dips can be observed in the graph, which has two causes; one is that the pie chart selection algorithm does not strictly select the highest fitness solution for procreation, it merely gives the higher fitness solutions higher likelihood of being selected; another cause is that mutation can in some cases lead to a lower fitness rather than a higher, as the purpose of mutation is to introduce randomness.

Similar results with a slightly quicker convergence were obtained using a slightly higher mutation rate of 5 %, as presented in Figure 6.2. However as the mutation rate grows, the solutions tend to deviate further from optimal with greater magnitude of the dips, see Figure 6.3 which presents the results obtained using a 6 % mutation rate. The deviances indicate that the mutations interfere with the algorithm’s learning.
CHAPTER 6. RESULTS AND VERIFICATION

6.1. INFLUENCE OF MUTATION RATE

Figure 6.1: Results obtained using a mutation rate of 3 %, population size of 10 ATC’s and 50 iterations to termination.

Conversely, a mutation rate which is too low also affects the results negatively, as a low mutation rate limits the reachability of the solutions as described in Section 2.3.3. This was confirmed through using a mutation rate of 1 % and the results are presented in Figure 6.4 which shows that the algorithm does converge towards optimal solutions, however requiring nearly 4 times more iterations than in the case where the mutation rate ranges between 3–6 %.

In Figure 6.5 the results for the case when the mutation rate is 0 % are presented. Note that the amount of iterations required to converge to optimal solutions differs marginally from the
6.1. **INFLUENCE OF MUTATION RATE**

**CHAPTER 6. RESULTS AND VERIFICATION**

Figure 6.3: Results obtained using a mutation rate of 6 %, population size of 10 ATC’s and 50 iterations to termination.

Figure 6.4: Results obtained using a mutation rate of 1 %, population size of 10 ATC’s and 50 iterations to termination.

Case when the mutation rate is 1 %. A mutation rate can likewise be too high, which results in difficulties for the algorithm to learn patterns of optimal solutions; this is illustrated in Figure 6.6 where the mutation rate was set to 100 %. Note that the solutions never converge to full fitness, despite the increased iterations of procreation, since the algorithm can not find a pattern for maximising the fitness of its solutions due to the continuous introduction of randomness.
CHAPTER 6. RESULTS AND VERIFICATION

6.1. INFLUENCE OF MUTATION RATE

Figure 6.5: Results obtained using a mutation rate of 0 %, population size of 10 ATC’s and 50 iterations to termination.

Figure 6.6: Results obtained using a mutation rate of 100 %, population size of 10 ATC’s and 100 iterations to termination.
6.1. Dynamic Mutation Rate

It was desired to increase the performance of the algorithm by optimising its learning rate. The graphs of Figures 6.1–6.6 suggest that mutations also may impair the learning rate, which is demonstrated by dips and ascends in the learning phase. Once convergence is achieved, mutations also cause deviation from full fitness, demonstrated as dips in the graphs of the aforementioned figures. It was therefore desired to study the effects on the performance of a dynamic mutation rate which decreases once the solutions have reached sufficient convergence to maximum fitness. An initial mutation rate of 10% was studied; the increment with which the mutation rate decreases was varied. For reference, Figure 6.7 presents results obtained with a static mutation rate of 10%.

Figure 6.7: Results obtained using a static mutation rate of 10%, population size of 10 ATC’s and 50 iterations to termination.

The mutation rate was then set to decrease dynamically with two different increments; Figure 6.8 presents the results obtained with a dynamic mutation rate with an initial value of 10% which decreased with 0.5% for every iteration once the solutions had reached 80% of the maximum conceivable fitness. Similarly, Figure 6.9 presents the results obtained where the mutation rate decreased with 1% for every iteration once the solutions had reached 80% of the maximum conceivable fitness. Through comparison of the three different cases, it is noted that the learning rate increases as the mutation rate is decreased once sufficient convergence to maximal fitness is achieved. When the mutation rate decreased with 1% rather than with 0.5%, the solutions converged quicker to maximal fitness; this suggests that it is beneficial to reduce the introduction of randomness to the data once sufficient convergence to maximal fitness is achieved; crossover of existing data is then sufficient to reach maximal fitness.

In order to optimise the algorithm, initial mutation rates ranging between 3–5% were considered rational, as they result in quick convergence and can be decreased rapidly. Different combinations of initial mutation rates and threshold percentages to maximal fitness were tested. Figure 6.10 presents results obtained using an initial mutation rate of 3% which was set to decrease with 0.5% for every iteration once the solutions had reached 70% of the maximum conceivable fitness; Figure 6.11 presents the same conditions however with a mutation rate which was set to decrease once the solutions had reached 80% of the maximum conceivable fitness. For reference, Figure 6.1 presents the same conditions however with a static mutation rate.
Comparing the three cases, it is evident that the learning rate is higher with a dynamically decreasing mutation rate than a static mutation rate. With a 70% threshold, the improvement in learning rate means that convergence to maximum fitness is reached in 9 iterations rather than 10; with a 80% threshold, the improvement means convergence in 7 iterations rather than 10.

Figure 6.12 presents the best results obtained using a 5% initial mutation rate which decreased with 0.5% once the solutions had reached 65% of the maximum conceivable fitness. For reference, Figure 6.12 presents the same conditions however with a static mutation rate. Comparing the two cases, the case with a dynamic mutation rate yields convergence to maximal fitness in 8 rather than 9 iterations as in the case with a static mutation rate. In all cases with dynamically decreasing mutation rates, less dips occur in the learning phase and no dips occur once maximum fitness is achieved due to the lack of mutations at that stage. Therefore it is concluded that the learning rate increases when the mutation rate decreases once the algorithm has learned sufficiently for convergence to the maximum conceivable fitness of its solutions. The best obtained result was with an initial mutation rate of 3% which decreased with 0.5% once the solutions had reached 80% of the maximum conceivable fitness.

Figure 6.8: Results obtained using an initial mutation rate of 10%, population size of 10 ATC’s and 50 iterations to termination. Mutation rate decreased with 0.5% every iteration once the fitness of the solutions had reached 80% of maximum conceivable fitness.
6.1. INFLUENCE OF MUTATION RATE

CHAPTER 6. RESULTS AND VERIFICATION

Figure 6.9: Results obtained using an initial mutation rate of 10 %, population size of 10 ATC’s and 50 iterations to termination. Mutation rate decreased with 1 % every iteration once the fitness of the solutions had reached 80 % of maximum conceivable fitness.

Figure 6.10: Results obtained using an initial mutation rate of 3 %, population size of 10 ATC’s and 50 iterations to termination. Mutation rate decreased with 0.5 % every iteration once the fitness of the solutions had reached 70 % of maximum conceivable fitness.
CHAPTER 6. RESULTS AND VERIFICATION

6.1. INFLUENCE OF MUTATION RATE

Figure 6.11: Results obtained using an initial mutation rate of 3 %, population size of 10 ATC’s and 50 iterations to termination. Mutation rate decreased with 0.5 % every iteration once the fitness of the solutions had reached 80 % of maximum conceivable fitness.

Figure 6.12: Results obtained using an initial mutation rate of 5 %, population size of 10 ATC’s and 50 iterations to termination. Mutation rate decreased with 0.5 % every iteration once the fitness of the solutions had reached 65 % of maximum conceivable fitness.
6.2 Influence of Populations

Based on the findings presented in Section 6.1, a fixed mutation rate of 3% and 100 iterations to termination were chosen, while population size and the amount of populations were varied.

6.2.1 Population Size

The population size was varied in steps of 10 ATC’s and the amount of iterations until the algorithm had learned patterns for optimal sets of solutions was compared. Figure 6.13 presents the results where the population size is 20 ATC’s; showing an approximate twofold of iterations required for the algorithm to converge to optimal solutions, compared to the case of a population size of 10, see Figure 6.1.

![Graph showing results with population size 20 ATC's](image)

Figure 6.13: Results obtained using a mutation rate of 3%, population size of 20 ATC’s and 100 iterations to termination.

Similarly, Figure 6.14 presents the results where the population size is 30 ATC’s and Figure 6.15 presents the results for a population size of 40 ATC’s. With 30 ATC’s, the best results require an approximate threefold of iterations more than the case of a population size of 10 ATC’s; with 40 ATC’s, the best result requires an approximate sixfold of iterations. The conclusion is that larger populations require more iterations for converging to optimal results, which is a reasonable conclusion, since a larger variety of ATC’s yield a larger variety of crossovers that can take place.
CHAPTER 6. RESULTS AND VERIFICATION  6.2. INFLUENCE OF POPULATIONS

6.2.2 Amount of Populations

The number of populations was varied and the amount of iterations until the algorithm had learned patterns for optimal sets of solutions was compared; the population size was set to 10 ATC’s and mutation rate was set to 3 %. Figure 6.16 presents the results with 10 populations and Figure 6.17 presents the results with 20 populations. The generated graphs using both 10 and 20 populations show insignificant deviations from that of Figure 6.1 which presents the
results with 5 populations. Thus it can be concluded that the amount of populations used does not result in significant changes in learning of patterns for optimal solutions.

Figure 6.16: Results obtained using a mutation rate of 3 %, population size of 10 ATC’s, 50 iterations to termination and 10 populations.

Figure 6.17: Results obtained using a mutation rate of 3 %, population size of 10 ATC’s, 50 iterations to termination and 20 populations.
6.3 Population Size and Mutation Rate

The effects of a greater population size and a varied mutation rate was studied in order to further the understanding of the system. The case where the population size was 40 ATC’s, using a mutation rate of 3 % and 100 iterations until termination was studied, see Figure 6.15; the performance of which was studied and improved upon through altering the mutation rate. Results obtained using the configuration of a dynamic mutation rate which was considered optimal in Section 6.1 is presented in Figure 6.18; that is an initial mutation rate of 3 % which decreases with 0.5 % for every iteration once 80 % of maximal fitness is achieved. It is evident that the learning rate increased with the dynamic mutation rate. Since the amount of iterations until termination is increased for the purpose of the larger population size, the increment with which the mutation rate decreases every iteration was altered; Figure 6.9 presents results obtained using an incremental decrease of 0.1 % once 65 % of maximal fitness was achieved, rather than 0.5 % at 80 %. The latter case yields a slight increase in performance compared to the case with smaller population sizes. For comparison, an initial mutation rate of 3 % with an incremental decrease of 0.5 % per iteration once 50 % of maximal fitness was achieved was tested and the results are presented in Figure 6.20. It is clear that limiting the mutation rate too early and too rapidly causes a decline in performance.

Figure 6.18: Results obtained using an initial mutation rate of 3 %, population size of 40 ATC’s and 100 iterations to termination. Mutation rate decreased with 0.5 % every iteration once the fitness of the solutions had reached 80 % of maximum conceivable fitness.
Figure 6.19: Results obtained using an initial mutation rate of 3 %, population size of 40 ATC’s and 100 iterations to termination. Mutation rate decreased with 0.1 % every iteration once the fitness of the solutions had reached 65 % of maximum conceivable fitness.

Figure 6.20: Results obtained using an initial mutation rate of 3 %, population size of 40 ATC’s and 100 iterations to termination. Mutation rate decreased with 0.5 % every iteration once the fitness of the solutions had reached 50 % of maximum conceivable fitness.
6.4 Discussion of Results

From the results presented above, it was concluded that the system is effective in increasing fitness of solutions through iteration. The performance of the machine learning algorithm proved to be strongly dependent on its parameters; the mutation rate proved to have greatest effect on the results and performance of the algorithm; it was demonstrated that delicate changes to the mutation rate can greatly influence the performance for the better or worse. It was also demonstrated that a moderate mutation rate increases the learning rate of the algorithm, as mutation introduces randomness in the data which increases the reachability of the solutions. It was concluded that a mutation rate ranging between 3–5 % was optimal; a lower mutation rate yields a low learning rate; a higher mutation rate yields difficulty in learning. A dynamic mutation rate which decreases as the total fitness score of solutions increases proved effective in increasing the learning rate through limiting the introduction of new data; this is beneficial since mutations yield unpredictable results for the fitness of solutions, thus limiting mutations when the fitness of solutions is sufficiently high increases the predictability which aids the algorithm in learning. The best result was obtained using an initial mutation rate of 3 %, based on the promising results using a static mutation rate of 3 %. The effect of a dynamic mutation rate was studied on larger populations, which proved effective in increasing the performance of the algorithm compared to the case where the mutation rate was static. However, optimising the selection of an initial mutation rate and threshold for decrease of mutation rate can be studied further.

It was desired to study whether the solutions which cause failures are identical or distinctive when the solutions had converged to maximum fitness. Through analysis of the logged solutions, as described in Section 5.1.2 it was apparent that in all cases where convergence to maximal fitness is achieved, the solutions are distinctive rather than identical. On the contrary, at the start of the iterative process, when maximal fitness is not achieved, the solutions are in most cases identical. This suggests that as more mutations occur and the algorithm discards more of the solutions which do not cause failures, the remaining solutions are those that cause failures; also the pool of solutions which cause failure increases with mutation.

The population size proved to influence the performance of the algorithm; a greater size requires more iterations of procreation for converging to maximal fitness of the solutions. This is a feasible conclusion as larger populations require more crossovers in order to achieve the same reachability as smaller populations. The amount of populations when varied resulted in insignificant changes to the performance of the algorithm. This is feasible since the size of the population does not change. The results in Section 6.3 suggest that the performance of the algorithm can be improved with large population sizes, through optimising the mutation rate.
Chapter 7

Conclusions and Future Work

This work depicts the design of a learning-based testing system for testing of safety-critical vehicle systems. The intention was to improve upon a testing tool with the ability to learn in order to reduce the time and cost of testing through automated generation and execution of test cases; however an alternative course was taken and instead of relying on the aforementioned testing tool to communicate with a target system, a target system was remodeled which facilitated direct communication with the learning-based system created in this work. A thorough background study was conducted on topics regarding model-based testing and machine learning. The background study laid the basis for the design of a machine learning algorithm and the target system to be tested, both of which constitute the learning-based testing system that was created in this work. The testing system was evaluated in terms of its ability to generate desired solutions, which in this application means test cases that result in failures; this is in order to discover the system’s defects. It is worthy to mention that a passed test case does not signify a lack of defects, but it does argue for an increased trust in the tested system’s correct functionality.

7.1 Conclusions

Based on the findings of this work, it is concluded that it is feasible to automate testing through outsourcing the work to machines with a learning ability; genetic algorithms proved to be successful for this cause. The findings are beneficial for the aforementioned AMASS project, the purpose of which solely is to reduce time and costs of assuring functionality of safety-critical systems in the automotive domain among others. The system created in this work may be applied to a variety of domains and applications without any changes to its architecture or its machine learning algorithm; this is because the system is modular, meaning that each function is contained in independent scripts which are imported to the main script. Two adjustments are required for applying the system to other applications:

1. A model of, or, another target system under test is required. Since the target system under test in this application is imported from an independent script, it is possible to interchange models of other systems through calling different model scripts.

2. For reading initial ATC’s, a problem specific parsing script is required to substitute the aforementioned readtables.py in Chapter 5. Similar to the model script, the parsing script is contained in an independent script.

7.2 Future Work

The fields of Artificial Intelligence and Machine Learning are vast and therefore rich of different types of algorithms, many of which can be applied to solve the same problems. It is therefore desirable to benchmark different algorithms that are feasible candidates for this application in
order to further this field and improve upon the performance of the current state of the art. Further, as mentioned in Section 6.4, it is desired to optimise the initial value of a dynamic mutation rate for a higher learning rate, as well as to optimise the threshold where the decrease of the mutation rate begins.

The system created for this work was intended to be an extension for a testing tool which facilitates communication with physical target systems; however due to the inability to reassemble the tool this was not conducted. For future advancements in this area, the findings of this work may be used to extend established testing tools with learning abilities. Further testing can be conducted on target systems of higher complexity in order to study the created system’s scalability. Furthermore, error location and error analysis are features which established testing software offer; these features are valuable as they aid in discovering the cause of errors. It is possible to extend the system created in this work with the aforementioned features; however it is regarded more feasible and cost effective to enhance existing, established, testing software with a learning ability.
Bibliography


Appendix A

A.1 Abstract test case

[SHORTNAME=atc4]

State:  
( GlobalBrakeController.idle ABSFL.idle )
GlobalBrakeController.x=0 ABSFL.x=0 GlobalBrakeController.rpm1=0 GlobalBrakeController .rpm2=0 GlobalBrakeController.reqTorque=0 GlobalBrakeController.whiTorque=0 GlobalBrakeController.v=0 GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4 =16 GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0 ABSFL.v =0 ABSFL.R=1

Transitions:  
GlobalBrakeController.idle->GlobalBrakeController.Entry { reqTorque := 0 , rpm1 := 8 , rpm2 := 0 , x := 0 }  

State:  
( GlobalBrakeController.Entry ABSFL.idle )
GlobalBrakeController.x=0 ABSFL.x=0 GlobalBrakeController.rpm1=8 GlobalBrakeController .rpm2=8 GlobalBrakeController.reqTorque=0 GlobalBrakeController.whiTorque=0 GlobalBrakeController.v=0 GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4 =16 GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0 ABSFL.v =0 ABSFL.R=1

Delay: 2

State:  
( GlobalBrakeController.Entry ABSFL.idle )
GlobalBrakeController.x=2 ABSFL.x=2 GlobalBrakeController.rpm1=8 GlobalBrakeController .rpm2=8 GlobalBrakeController.reqTorque=0 GlobalBrakeController.whiTorque=0 GlobalBrakeController.v=0 GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4 =16 GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0 ABSFL.v =0 ABSFL.R=1

Transitions:  
GlobalBrakeController.Entry->GlobalBrakeController.Reaction { x >= 2 , v := (rpm1 + rpm2 + rpm3 + rpm4) * R / 8 , whiTorque := reqTorque / 4 }  

State:  
( GlobalBrakeController.Reaction ABSFL.idle C1.idle C4.exec _JRGENT.S )
GlobalBrakeController.x=2 ABSFL.x=2 GlobalBrakeController.rpm1=8 GlobalBrakeController .rpm2=8 GlobalBrakeController.reqTorque=0 GlobalBrakeController.whiTorque=0 GlobalBrakeController.v=6 GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4 =16 GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0 ABSFL.v =0 ABSFL.R=1

Transitions:  
GlobalBrakeController.Reaction->GlobalBrakeController.idle { }  

State:  
( GlobalBrakeController.idle ABSFL.idle )
GlobalBrakeController.x=2 ABSFL.x=2 GlobalBrakeController.rpm1=8 GlobalBrakeController .rpm2=8 GlobalBrakeController.reqTorque=0 GlobalBrakeController.whiTorque=0 GlobalBrakeController.v=6 GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4 =16 GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0 ABSFL.v =0 ABSFL.R=1
A.2 Information extracted from ATC

Tables extracted from the ATC presented in Appendix A. Note that the only a part of the complete file is presented here, however the remainder of the file follows the same pattern.