UNIVERSITÄT DES SAARLANDES

Saarland University

Master Thesis

AUTOMATIC TEST CASES GENERATION AND SIMULATION FROM VEHICLE CRASH REPORTS

Author:
Tri Huynh

Supervisor:
Prof. Dr.-Ing. Andreas Zeller

Advisor:
Dr. Alessio Gambi

Monday 18th September, 2017

Abstract

Finding test cases which reveal when autonomous cars malfunction is a challenging task because the number of possible test cases are infinite. Additionally, it is difficult to find factors that can unveil these safety bugs. Therefore, one method to critically test the autonomous cars' safety is to simulate real life crash scenarios, and test autonomous cars in virtual environments. With this motivation, I propose a novel approach which uses Natural Language Processing to analyze crash descriptions, extract environment properties and storyline of the accident, and consequently generate test cases. The generated test cases contain precise and adequate information for simulation tools to reconstruct the crash environment and the story of the accident. Furthermore, the proposed system simulates these driving scenarios using a state-of-art rendering engine to demonstrate that the testers' simulators can be automatically configured to execute the generated tests, such that autonomous car implementations can be directly tested in relevant and critical scenarios. Having such system and test cases will help testers in understanding how their design will behave in critical situations that lead to crashes and to check if crashes could have been avoided using self-driving cars.

1 Introduction

Autonomous cars have been demonstrating their excellent capability in avoiding traffic accidents and in mitigating their criticality. However, a few reported crashes [12][27] show that autonomous cars can perform worse than human drivers, leading to avoidable accidents in certain scenarios. For example [12], a Google autonomous car hit a bus while switching lane because the car assumed that the bus would stop and let the car move into the lane first. If the car was driven by a human driver, they would let the bus pass by completely first and after that proceeded to the lane to prevent the accident from happening. Finding cases which reveal the flaws of autonomous cars is a tremendously challenging task. Firstly, the number of possible test cases to be covered are infinite because autonomous cars sensors and actuators are expected to safely operate on different road types, terrains, under various lighting and weather conditions [3][8][9]. Importantly, autonomous car must deal with unpredictable movements of surrounding objects like other vehicles, pedestrians, and animals. Secondly, detecting or predicting extraordinary situations which unveils that autonomous cars can perform worse than ordinary human drivers are extremely difficult given the limitation of human capability and time budget. Therefore, one solution to tackle the two problems is to test autonomous cars in simulated motor vehicle accidents [2][10][11]. If autonomous cars can perform better than human drivers in these vehicle crashes by mitigating or avoiding damage, then autonomous cars can be considered as safer and better than human drivers in these particular accidents.

With this motivation, I propose a novel approach which semi-automatically takes a set of crash reports as input, i.e., testers manually input selected crash reports, then the system automatically generates corresponding test cases. For each crash report, Natural Language Processing (NLP) and an Ontology with concepts related to car crashes are applied to automatically generate test cases that accurately reproduce the reported crash (original test case), or modified versions from the original crash (derived test cases). Each of the test case contains the properties of the surrounding environment, the initial positions and movement directions of each involved vehicle(s) or pedestrian(s), as well as the storyline of the accidents. The *original test cases* aim to not only save the tester effort in manually creating the test case for a given accident, but also allow testers to build effective and focused test suites to assess the quality of different autonomous cars implementations. Besides the original test cases, which reproduce the story of a given accident, the *derived test cases* are generated based on predefined criteria, such as harsh weather conditions, the presence or absence of obtrusive objects from the original crash scenario. The derived test cases aim to create various test setups for testers to test their autonomous cars in slightly different environments based on the tester's purposes.

The original and derived test cases are created in both textual and structural formats. Although textual test cases give testers a descriptive storyline of the corresponding accident, they do not help testers in verifying the behaviors of their autonomous cars in the simulation of the crash. For this reason, I will use a 3D Simulation software named BeamNG [29] to reconstruct and simulate the crash scenario in a virtual world. Specifically, I include all necessary information about the environment and actors of each test case, which has its own BeamNG object file. Then the object files are used to construct 3D simulations of the accidents in BeamNG, a state-of-art rendering engine. This process aims to demonstrate that the test cases can be read as inputs into the testers' simulation system, which contains the simulated model of the autonomous car, and 3D world generation features to create objects needed for reconstructing the environment of a given accident. By examining these 3D simulations of the autonomous cars in the reconstructed accidents, the testers can verify whether their autonomous cars behave appropriately according to the testers' expectation and the cars' requirements.

One can apply another tools or frameworks to implement the reconstruction of crash scenario in 3D Virtual World. For example, one can construct the test cases using OpenDRIVE [7] or OpenStreetMap (OSM) [6], a powerful and comprehensible format to specify the environment properties and location of objects. Then one can simulate the test case in Unity with Road and Car generator frameworks like RoadArchitect [4] and Edy's Vehicle Physics [28]. However, when compared to these file formats and frameworks, BeamNG offers numerous considerable advantanges:

1. **Simple Object Constructor**: One can create road, car, movement path, and lighting condition by simply using the corresponding object constructor in an existing BeamNG scenario. Then placing these object constructors in one file and BeamNG will render the objects accordingly.

2. **Simple Object Customization**: One can modify the object properties, such as the width, pattern, and grade of the road, or the model and color of a car by simply specifying the name of the material or color. Furthermore, the position of an object like a vehicle, waypoint (a point in the movement path of a vehicle), or a road node can be specified by giving the horizontal, vertical, and depth coordinations.

3. **Customizable Built-in AI**: BeamNG provides a Lua interface for configuring the speed limit and movement style of vehicles e.g. following waypoints on the road or chasing another vehicle. This feature saves a remarkable effort in developing a mechanism to ensure that vehicles travel along its path only on the road under a particular speed constraint, and will eventually crash into an object.
4. **Built-in Slow Motion, different Camera Views, and Video Recording features**: One can adjust how fast the crash happens and place the camera at a specific angle and position to observe the accident and behaviors of a vehicle at a particular time. Furthermore, one can record a video for conducting user study or future reviews on the chosen crashes. All the built-in features can be triggered only by pressing the corresponding keys or selecting the option in BeamNG GUI interface.
5. **Detailed Physics Simulation**: BeamNG simulates the damage of a car by the impact location and force. Therefore, the damages to a vehicle are reflected precisely on the impacted components.

In summary, my thesis will contribute an NLP method to generate test cases for testing autonomous cars by extracting the storyline as well as properties of the environment and involved objects from actual car crash reports. Consequently, the result of the thesis is expected to provide an approach to critically test the safety of autonomous cars by verifying the behaviors of simulated autonomous cars in the 3D reconstruction of reported traffic accidents, which are recreated from the generated test cases.

2 Technical Background and Related Work

This section briefly described background concepts used throughout the thesis, and presents the related work in these fields.

2.1 Technical Background

2.1.1 Ontology

Ontology is a representation of concepts to model a domain of knowledge [15]. An ontology consists of classes (concepts), the properties (aka. slots or roles) to express the attributes and features of each class, and the restrictions on these properties (aka. facets or role restrictions) [16]. Additionally, a class can contain subclasses to provide a more detailed description of the superclass. For example, a class of cars represents all the types of car in the world, each car has several properties such as car body (minivan, hatch-back, etc.), fuel, manufacturers, and so on. Regarding role restrictions, a car can have only one car body style, but it can be powered my more than one type of fuel, like hybrid cars.

2.1.2 Word2Vec Concept Matching

Word2Vec [27] is a machine learning method which takes a corpus text as input and outputs word vectors. First, the algorithm constructs vocabulary from the text and learns the vector representation of these words. For example, the vector representation of countries in the world are:

germany	0.563291
spain	0.678515
belgium	0.678515
netherlands	0.678515
italy	0.678515
luxembourg	0.610033
portugal	0.577154
switzerland	0.678515

From this word vectors, one of the applications is finding whether a word belongs to a domain by looking for all the words from a range. For instance, from the above example, one can define the Country domain has word vectors value from 0.5 to 0.7, hence a country name like "Germany" is considered as a word in the "Country" domain.

2.1.3 Natural Language Processing (NLP)

Natural Language Processing is the ability of computational technologies to process human natural language. The process can be done in automatic or semi-automatic manners [17].

2.1.4 Part of Speech (POS) Tagging

Part of Speech Tagging is an NLP algorithm which "assigning part-of-speech marker to each word in an input text" [18]. POS refers to the word class of a word. In English, there are 8 POS: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article.

For example, consider the following sentence:

Vehicle1 turned left and was struck on the right side by the front of Vehicle2

After using POS Tagging, the result is:

Vehicle1_NN turned_VBD left_RB and_CC was_VBD struck_VBN on_IN the_DT right_JJ side_NN by_IN the_DT front_NN of_IN Vehicle2_NN

The interesting tags in the scope of the Thesis are NN (Nouns), all VBX (Verbs) tags where X is a character specifying the Verb form (gerund, infinitive, past, etc.), RB (adverb), JJ (adjective). The description of all tags are presented in the Penn Treebank by Marcus et al. [26].

2.2 Related Work

My system will apply NLP POS Tagging algorithm to identify mostly the verbs, nouns, and adjective from the crash scenario descriptions, then by applying the knowledge base in the Ontology, the system will be able to extract the properties of the environment and actors in a particular accident, as well as the storyline of the crash. Afterward, these information will be used to generate test cases, which will be provided as input to the simulation tools of the autonomous car manufacturer. Next, the tool constructs the virtual 3D environment of the crash scenario, the model of autonomous cars, and simulate the behaviors of the cars in the given accident.

This section discusses the related work of the proposed approach above.

2.2.1 Generating Test Cases Using NLP

NLP has been applied to generate test cases for computing systems [19][20][22]. In these systems, the test cases are generated from a structured software requirement specifications, with each requirement being written using a dedicated syntax. However, in my system, the test cases are generated from crash report descriptions, despite that the report is recorded using natural language without following any predefined syntax.

Sippl et al. [1] proposed a method to generate test cases for autonomous cars from the data of driving simulation and requirements of autonomous cars. In terms of input, my system also extracts the properties to reconstruct the test environment, but the input source in my system is from actual crash reports, which contain at least the description of the crash scenario, not from simulated environment. Furthermore, the format and content of my system's test cases are similar to their proposed system, but my system will contain weather, lighting conditions as extra information.

Masuda [3] generate test cases based on different sensors on autonomous cars, and their possible values, then he simulates the test cases in virtual environment and display the test results. Similarly, my system also generates test cases, but based on actual crash reports. Furthermore, the simulation of my system will not output expected result because this will be decided by the testing and development teams for each selected accident. The purpose of my system's simulation is to demonstrate that the test cases of recorded traffic accidents can be inputted into the testers' simulator, so the testers will be able to evaluate the performance of their virtual autonomous car models in the simulated crash scenarios.

2.2.2 Application of Ontology in autonomous car or ADAS (Advanced Driver Assistance System) development

Armand et al. [21] applied Ontology to construct a knowledge base for ADAS to identify the context of the road, how the perceived entities on the road are expected to behave, and what are the consequences of these behaviors, thus improving the context awareness ability of the ADAS system. Additionally, Zhao et al. [23] developed an Ontology for ADAS containing knowledge base about road map, traffic regulations, vehicle properties and controls. In these research work, their knowledge bases which specifies the possible actions and properties of vehicles traveling on the street can be applied to my system in order to specify the actions and properties of involved actors in the accidents.

2.2.3 Real-time Autonomous Cars/ADAS Testing in Simulated 3D Environment

Erbsmehl [2] developed a system to simulate the behavior of a car with or without a safety system, given a crash scenario being retrieved from GIDAS (German In-Depth Accident Study) database. In my system, I also examine real crash reports source to generate test cases, and simulate the accidents in 3D environment. However, my system aims at generating test cases from real vehicle crashes, not at the behaviors of autonomous car in the virtual environment, so I will not implement the simulation for the safety system, sensors, or actuators of autonomous car in my Thesis.

Furthermore, there exists research work [9][11][24] that test autonomous cars/ADAS in simulated 3D environment. In these researches, the components of autonomous cars/ADASs are modeled using 3D simulation systems. Then a number of traffic scenarios are fed into the simulator, and the autonomous cars/ADASs behaviors and features are tested under the given scenarios. My system is proposed to support this approach, by giving test scenarios for recorded traffic accidents into these 3D simulation platforms. As a result, testers are able to verify whether their autonomous car/ADAS models can avoid accidents in the given crash scenarios.

3 Method

The test cases generation process for an accident being recorded in the crash database can be summarized in Figure 1 below:

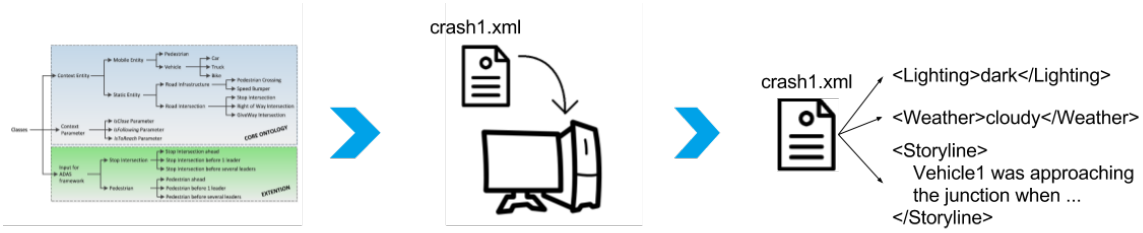
3.1 Build the Traffic Accident Ontology

The Ontology is constructed to store the concepts related to crashes involving pedestrians such as Weather, Lighting, Obtrusive Crash properties, Unobtrusive Crash properties, Vehicle actions (turn left, right, U-turn, etc.), Pedestrian actions (cross the road, walk, etc.), and so on. This will help the system identifies the environment properties and the actions of each actor (vehicle or pedestrian) in the accident. When combining with the NLP Part-of-Speech (POS) algorithm, the system will be able to match a word, which could be a noun, adjective or verb, into the corresponding concept being defined in the Ontology.

For example, if the crash scenario description text contains the word “rain”. Then the POS algorithm will tag the String “rain” as Noun, and look up in each concept of the Ontology, then the system finds that “rain” belongs to the Weather concept, so the system concludes that the accident occurred under a rainy weather.

Proposed Tool and Implementation: The Ontology will be constructed using Protege tool and then export to OWL/RDF format. Firstly, each factor contributing to traffic accidents will be declared as a Class in the Ontology. Next, for each factor, all the possible values are defined as subclasses. Finally, the complete Ontology is incrementally constructed with related concepts to the selected traffic accidents. The Ontology is always read when the system starts so that another crash report can be read without reading the Ontology file again.

For instance, let "Weather" be an accident contributing factor, then "Rainy", "Sunny" and "Cloudy" can be declared as subClasses of "Weather" to represent different possible forms of weather.



1.Build Traffic Accident Ontology

2. Read a Crash Report

3. Extract Necessary Information

Weather	Lighting	Lane Number	...
⦿ Sunny	O Dark	O 2	O ...
O Cloudy	⦿ Bright	⦿ 4	⦿ ...
...	...	...	...

4. Select Properties of Derived Test Cases

Inputs	Steps to Reproduce
Weather: cloudy **Lighting**: dark **Initial Position**: + Car 1: [-20, 30, 0] + Car 2: [20, 40, 0] ...	1. Car 1 approaches junction 2. Car 2 suddenly turns at junction 3. Car 1 does not brake on time, two cars collide

5a. Generate Original Textual Test Case

Inputs	Steps to Reproduce
Weather: sunny **Lighting**: bright **Initial Position**: + Car 1: [-20, 30, 0] + Car 2: [20, 40, 0] ...	1. Car 1 approaches junction 2. Car 2 suddenly turns at junction 3. Car 1 does not brake on time, two cars collide

5b. Generate Derived Textual Test Cases

crash1.prefab

```
...
new ScatterSky(crash1_Sky) {
   ...
   skyBrightness = “200”; // cloudy sky
   brightness      = “-1”; // dark
};

new BeamNGVehicle(car1) {
   ...
  position = "-20 30 0";
 };
...
```

6a. Construct Original Test Case using BeamNG syntax

crash1derived.prefab

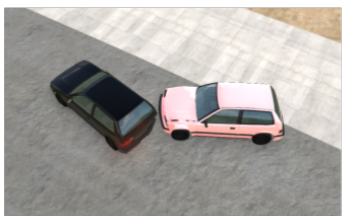
```
...
new ScatterSky(crash1_Sky) {
   ...
   skyBrightness = “350”; // sunny sky
   brightness      = “1”; // bright
};

new BeamNGVehicle(car1) {
   ...
  position = "-20 30 0";
 };
...
```

6b. Construct Derived Test Cases using BeamNG syntax

crash1.prefab / crash1derived.prefab

BeamNG



7. Run the Test Case in BeamNG

Figure 1: Proposed Test Case Generation Process

3.2 Read a Crash Report

The environment properties and story of each accident are stored in a crash report, which usually are publicly available in structured formats like XML or CSV because they are extracted from a database. Therefore, in order to extract the environment and actors properties, as well as the sequence of actions in a given accident, the system needs to parse the crash report file first, then it retrieves all the XML tags that contain the information about the storyline, environment properties, and involved actor(s) properties of the given accident.

Proposed Tool and Implementation: My system will use an XML (or CSV) parser to read the content of several crash reports in XML format [5]. Then from the parsed content, the system extracts the storyline of the accident and called the NLP POS algorithm to analyze the crash scenario. The system finds the properties of the actors and environment in the crash description, if the information can not be found, then the system attempts to extract the data from other XML attributes of the crash report. For example, consider the following XML crash report:

```
<CrashReport>
        <Lighting>Bright</Lighting>
        <Atmospheric_Condition>Clear</Atmospheric_Condition>
        <Summary>
            V1, a 2007 Acura MDX, was traveling west on the roadway
            negotiating the curve to the left on a downgrade and approaching
            a "T" intersection to the left. V2, a 2008 Land Rover L2, was
            traveling east on the same roadway with an upgrade ...
        </Summary>
</CrashReport>
```

Since the light condition is not given in the <Summary>section, the system will parse the XML file of the crash report, find the tags <Lighting>, <Atmospheric_Condition> (Weather). Then the system extracts the corresponding values which are "Bright", "Clear", and the storyline inside the tags.

3.3 Extract the Properties of Environment and Actors, along with the Storyline of the Accident

Because the crash reports in NHTSA database always contain the development of the accidents, the storyline for each crash report can always be found when the system reads the crash reports. After that, the system will apply POS algorithm and Ontology to analyze the sequence of actions in the given accident, and extract the three groups of data from the crash description. In case an accident contributing factor (e.g. Weather, Lighting) cannot be found from the storyline, nor from the XML or CSV crash reports, the system will assign a predefined value to the factor.

Proposed Tool and Implementation: The system can use Stanford NLP POS Tagger [13] and Porter Stemmer [14] tools to execute this step as follows:

1. Stanford POS Tagger tool is called to determine the nouns, adjectives and verbs in the given crash description.
2. For each noun, adjective and verb, a Porter Stemmer tool is applied to stem the word.
3. The stemmed word will be queried against the Ontology to find whether the word is a substring of a concept which belongs to the properties of environment or involving actors, the storyline of the accident, or none of them.
4. Store the properties of environment and involving actor(s) as key-value pair in a dictionary called Crash Properties Dictionary. For example: [“Weather”: “rain”]
5. Construct the steps to reproduce the accident by assigning stemmed verb(s) to refered noun(s) for each sentence in the storyline. Each action is stored as a string in a String array. The order of the sentences is also the chronological order of the events in the accident i.e. the first sentence describes what happens first, the second sentence describes what happens next, and so on.

3.4 Select Properties for Derived Test Cases (Optional)

If testers need to generate modified versions of the original test cases, the system presents a GUI for testers to select or define the values for each property of environment and involved actors, as well as adjusting the movement speed of a particular actor, or modifying the presence of obtrusive object(s) in the scenario. To help testers identify the crash attributes of the original test case, the system reads the Crash Properties Dictionary to find the properties of involved actors and environment, and highlight the values of the corresponding property in the GUI. As a result, testers can identify the crash attributes which are not in the original test case, and decide whether to select these attributes in the derived test case.

Since generating derived test cases is only necessary when testers need to create different scenarios from the original crash, this feature can be skipped if testers only focus on simulating the original crash scenario.

Proposed Implementation: I will extract the name of each environment property as well as its possible value by reading the Ontology. Specifically, the system will look into the "Environment Properties" Class and retrieve all the nearest subclasses, which are the related concepts of the environment property (Weather, Lighting, etc.). Next, for each of these subclasses, all of its child classes are the possible values of a given environment property. As a result, the system groups all the values belong to a given environment property and display them on the screen.

The properties of involved actors, and obtrusive object(s) addition/deletion will be modified directly in the corresponding text field. For example, if the tester wants to adjust the initial speed, they will change the value of the 'Initial Speed' text fields.

3.5 Generate Textual Test Cases (Optional)

The textual test cases aim to provide an overview of the crash scenario. To generate a test case, the system reads the Crash Properties Dictionary and extracted storyline to define the value for inputs and reproduce steps, then display the result.

Proposed Implementation: The input of a test case can be constructed by displaying each key-value pair in the Crash Properties Dictionary. The reproduce steps can be constructed by reading each String in the storyline array. Finally, the system presents the inputs of the test case and reproduce steps to the testers.

3.6 Generate Test Cases in BeamNG format

To provide a structured input into a simulator, the system organizes the inputs and reproduce steps according to the BeamNG formats of Road, Vehicle, or Waypoint objects, so that they can be parsed by BeamNG to construct the environment properties and actors' movement.

The syntax of the BeamNG object constructor offer a convenient and comprehensible interface to store the properties of the environment and involved objects, as well as the movement paths which are required to reconstruct the accident.

Furthermore, with BeamNG software, I can also apply more realistic environment reconstruction in the future by including different type of objects like buildings, trees and animals in a specific location inside the simulated environment. This will help testers understand the background of the accidents or infer related factors that contribute to the accidents from the surrounding environment.

Proposed Tool and Implementation: I will copy the templates of the BeamNG Road, Vehicle, and Waypoints object constructors, and modify the properties of environment and involved actors in the correspondding object constructor e.g I can specify the color of a car by modifying the color attribute in the Vehicle object as 'color = "Red";'. To construct the crash scenario in a step-by-step manner, I will extract the actions of each actor in each step from the storyline array, and construct the trajectory of the actor based on the action. For example, if Car1 goes forward, then I will construct a movement path of Car1 from [x, y, z] to [x + 20, y, z]. The turn and U-turn trajectory is constructed with multiple turning points. Next, from the trajectory of the involved actors and the storyline, the system determines a collision point where the main actors collide. As a result, I will have a collection of points in the trajectory of each actor from the starting position to the collision

point. Each point will be stored in a Waypoint Object Constructor. Finally, the properties of environment and involved actors, as well as the trajectory of each actor are read by BeamNG to reconstruct the accident.

3.7 Run the Test Cases in BeamNG

The test cases can be read by BeamNG to reconstruct the crash scenario. Specifically, BeamNG parses the object file of the scenario, then it extracts the objects which store the properties of environment and involved actors, as well as the movement path of each actor. Next, BeamNG will generate the 3D world with streets, weather, lighting conditions, and setup the position and properties of actors in the scenario based on the extracted information from the object file.

Proposed Tool and Implementation: I will create template files to store the attributes of Lighting, Vehicles, Streets, and Waypoints objects according to BeamNG syntax. Next, I will generate objects based on the template, and modify certain attributes in the objects to create a crash scenario similar to the description of the accident in the crash report. Finally, all the objects are stored in a single file. In order to play the scenario, one can open BeamNG and select the scenario, then BeamNG will construct the crash scenario by reading the position, movement path, 3D Model, etc. of each actor and other environment properties in the scenario's object file. After the scenario construction is finished, one can play the scenario, adjust the camera, and modify the objects in the scenario to suit their needs.

4 Evaluation

The evaluation of the system is organized into two parts: qualitative study and quantitative analysis. To prepare the data set for evaluation, the system will generate test cases from 10 to 20 recorded accidents from the National Motor Vehicle Crash Causation Survey database of National Highway Traffic Safety Administration of United States [5]. These crash reports will be picked randomly across different accident categories to demonstrate that the system can generate test cases from various crash reports.

4.1 Qualitative Study

In the qualitative study, a small number of test subjects (from 7 to 11 people) will be given a task to assess the similarity between the original crash report and the crash scenario simulation, as well as the textual description of the generated test cases. Specifically, I will select 3 random accident descriptions for each participant, then for each accident, I present the textual form and 3D simulations of the original and one derived test cases. After that, I will ask the participants questions to evaluate the similarity of the original test case to the corresponding accident description, and whether the derived test case is generated accurately according to the modified attributes. For each question, the participants grade the accuracy of the test case by selecting one out of four options "Exactly the same", "Similar", "Different", "Totally Different" with the scores being assigned for the options are 4, 3, 2, and 1 respectively.

In the end, all the answers for each question is collected, then I calculate the average score for each answer. The score will be interpreted as follow:

Table 1: Grading Scheme for Test Case Accuracy

Average Score	**Accuracy**
above 3.0	All the generated test cases are similar to the original crash description
from 2.5 to 3.0	On average, the generated test cases are quite similar to the original crash description
below 2.5	The generated test cases are considerably dissimilar to the original crash description

4.2 Quantitative Analysis

In terms of quantitative analysis, for each selected accidents, I will verify whether the generated test case has a similar pattern to its original description by analyzing the trajectory of each involved actors to infer the crash type (rear-end collision, forward impact, etc.) in the corresponding crash report. The trajectory can be inferred by checking the initial and collision positions of each actor in Unity. If the collision type being inferred from the trajectories of all main actors matches the crash type being specified in the crash report, then the simulation is accurate. In the end, the percentage of accurate test cases determine the accuracy of the system in simulating the accidents.

5 Schedule

The Thesis milestones are estimated given a duration of 26 weeks (180 days). The planned Thesis registration time is on 12 February 2018. The expected completion date is on 12 August 2018.

Each milestone in the schedule already includes the time to document the implementation step in the Thesis.

Table 2: Thesis Schedule

Weeks	Task
1	Implementing Ontology for defining concepts related to general traffic accidents
2-6	Read a crash report and extract the properties of environment and actors, as well as the procedure of the accident
7-8	Export the information in the test case into a BeamNG object file
9-12	Simulate the object file in BeamNG
13-14	Generate the derived test cases, and export them into textual format
14-19	Run the system to generate original and derived test cases for at least 10 accidents. Evaluate the result and refine the Ontology, NLP algorithm, Test case generation, and 3D simulation if necessary.
20-21	Conduct qualitative study and document the participants' evaluation in the Thesis
22-24	Refine and finalize the Thesis
25-26	Contingency time budget of 2 weeks

6 Success criteria

The system being developed in this Thesis shall satisfy all Must-Have criteria to be considered as successful:

Feature 1: The system shall apply an NLP algorithm to extract the storyline of the accident.

Feature 2: The system shall extract the properties of environment, involved vehicle(s), and pedestrian(s) from a crash report.

Feature 3: The system shall be able to store the properties of the environment, the properties of the involved vehicle(s) and actor(s), as well as the storyline of the accident in a file according to the BeamNG format. This file is considered as the original test case of the given crash report.

Feature 4: The system shall be able to simulate the accident from the corresponding constructed BeamNG object file using the BeamNG software. The simulation shall contain at least the 3D models of street, actor(s) (vehicle or pedestrian), as well as the movements and positions of the involved actor(s) . These movements and positions shall match with the corresponding crash description of the accident.

Feature 5: The system shall be able to simulate at least 10 accidents from two crash types e.g Rear End Collision, Forward Impact. In other word, for each of the two crash types, the system shall be able to reconstruct at least 5 accidents

Additionally, the system should also implement certain May-Have features to contribute more value to the testers in testing autonomous cars.

Feature 6: The system should be able to generate a test case in human-readable format, which reproduce the given crash scenario from the crash report database. The test case should contain the following information:

1. **Inputs**: Weather and Lighting Condition, the Initial Position, Moving Direction, and Traveling Speed of each vehicle and pedestrian involving in the accident.
2. **Reproducible steps:** The sequence of actions needed to be done in order to reconstruct the test case. This sequence is extracted and reconstructed from the crash description of the corresponding accident.

Feature 7: The system should be able to generate derived test cases from the one being generated as described in Feature 3. The derived test cases are created in a customized manner, which testers need to specify one or more of the following attributes to the system:

1. **Traveling Speed:** Adjust the movement speed of a particular actor in the scenario. The speed shall not exceed 200 km/h.
2. **Obtrusive / Unobtrusive Scenario:** The tester can modify the presence of an obtrusive object, and specify the coordinator of the object in the map.
3. **Weather Condition:** The crash happens under rain, cloudy, or sunny weather.
4. **Road Lanes:** The number of lanes on the road, which ranges from 1 to 3 for each direction.

Feature 8: The system shall be able to generate test cases for accidents of another crash type that are not covered in Feature 5. For example, Intersecting Paths or Same Trafficway Opposite Directions crash type.

Finally, the system will not cover a number of criteria which are out of its scope.

Must Not Have Requirement 9: The system will not support play backward / forward functions when simulating the reconstructed crash scenario

Must Not Have Requirement 10: The system will not simulate the sensors and actuators of an autonomous car in the crash scenario simulation

Must Not Have Requirement 11: The system will not be required to run in different Linux OSs, or Mac OS.

Table 3: Summary of the Expected Thesis Features

Feature	Must-Have	May-Have	Must-Not Have
Feature 1	xx	–	–
Feature 2	xx	–	–
Feature 3	xx	–	–
Feature 4	xx	–	–
Feature 5	xx	–	–
Feature 6	–	xx	–
Feature 7	–	xx	–
Feature 8	–	xx	–
Requirement 9	–	–	xx
Requirement 10	–	–	xx
Requirement 11	–	–	xx

References

[1] Christoph Sippl, Florian Bock, David Wittmann, Harald Altinger, and Reinhard German. *From Simulation Data to Test Cases for Fully Automated Driving and ADAS*, pages 191–206. Springer International Publishing, Cham, 2016.

[2] Christian Erbsmehl. Simulation of real crashes as a method for estimating the potential benefits of advanced safety technologies. In *Proceedings of the 21st international technical conference on the enhanced safety of vehicles (ESV 2009)*, number 09-0162, 2009.

[3] S. Masuda. Software testing design techniques used in automated vehicle simulations. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 300–303, March 2017.

[4] MicroGSD. Roadarchitect plugin in unity. Last Viewed: 26 August 2017, https://github.com/MicroGSD/RoadArchitect.

[5] National Highway Traffic Safety Administration. National motor vehicle crash causation survey. https://www-nass.nhtsa.dot.gov/nass/nmvccs/SearchForm.aspx.

[6] OpenStreetMap. Osm xml, 2017. http://wiki.openstreetmap.org/wiki/OSM_XML.

[7] VIRES Simulationstechnologie GmbH. Opendrive homepage, 2006. http://www.opendrive.org/index.html.

[8] M. Nentwig and M. Stamminger. Hardware-in-the-loop testing of computer vision based driver assistance systems. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 339–344, June 2011.

[9] M. R. Zofka, S. Klemm, F. Kuhnt, T. Schamm, and J. M. Zöllner. Testing and validating high level components for automated driving: simulation framework for traffic scenarios. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 144–150, June 2016.

[10] P. Minnerup and A. Knoll. Testing autonomous driving systems against sensor and actuator error combinations. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 561–566, June 2014.

[11] D. Gruyer, S. Choi, C. Boussard, and B. d'Andréa Novel. From virtual to reality, how to prototype, test and evaluate new adas: Application to automatic car parking. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 261–267, June 2014.

[12] California Department of Motor Vehicle. Google autonomous car crash report on february 14, 2016. https://www.dmv.ca.gov/portal/wcm/connect/3946fbb8-e04e-4d52-8f80-b33948df34b2/Google_021416.pdf?MOD=AJPERES.

[13] The Stanford NLP Group. Stanford nlp pos tagger homepage, n.d. https://nlp.stanford.edu/software/tagger.shtml.

[14] M. Porter. Porter stemmer homepage, 2006. https://tartarus.org/martin/PorterStemmer.

[15] T. Gruber. Ontology, 2009. http://tomgruber.org/writing/ontology-definition-2007.htm.

[16] D. L. McGuinness N. F. Noy. Ontology development 101: A guide to creating your first ontology, n.d. Stanford University, http://tomgruber.org/writing/ontology-definition-2007.htm.

[17] J. Thanaki. Python natural language processing, 2017. Packt Publishing, http://proquest.safaribooksonline.com.ezproxy.lib.rmit.edu.au/book/programming/python/9781787121423?bookview=overview.

[18] D. Jurafskyi and J. H. Martin. Speech and language processing, 2017. Chapter 10 Part-of-Speech Tagging, Stanford University, Stanford, USA, https://web.stanford.edu/ jurafsky/slp3/10.pdf.

[19] A. Ansari, M. B. Shagufta, A. Sadaf Fatima, and S. Tehreem. Constructing test cases using natural language processing. In *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, pages 95–99, Feb 2017.

[20] Gustavo Carvalho, Diogo Falcão, Flávia Barros, Augusto Sampaio, Alexandre Mota, Leonardo Motta, and Mark Blackburn. Test case generation from natural language requirements based on scr specifications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1217–1222, New York, NY, USA, 2013. ACM.

[21] A. Armand, D. Filliat, and J. Ibañez-Guzman. Ontology-based context awareness for driving assistance systems. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 227–233, June 2014.

[22] Prerana Pradeepkumar Rane. *Automatic Generation of Test Cases for Agile using Natural Language Processing*. PhD thesis, Virginia Tech, 2017.

[23] L. Zhao, R. Ichise, T. Yoshikawa, T. Naito, T. Kakinami, and Y. Sasaki. Ontology-based decision making on uncontrolled intersections and narrow roads. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 83–88, June 2015.

[24] C. Berger, M. Chaudron, R. Heldal, O. Landsiedel, and E. M. Schiller. Model-based, composable simulation for the development of autonomous miniature vehicles. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, DEVS 13, pages 17:1–17:8, San Diego, CA, USA, 2013. Society for Computer Simulation International.

[25] Unity. Unity homepage, n.d. https://unity3d.com/.

[26] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[27] B. Vlasic and NEAL E. Boudette. Self-driving tesla was involved in fatal crash, u.s. says, 2016. https://www.nytimes.com/2016/07/01/business/self-driving-tesla-fatal-crash-investigation.html.

[28] A. Garcia. Edy's vehicle physics homepage, n.d. http://www.edy.es/dev/vehicle-physics/.

[29] BeamNG. Beamng homepage, n.d. https://beamng.gmbh/.