Department of Computer Science

Investigating the use of OpenScenario in Developing Tests for Simulation-based Verification of Autonomous Driving Functions Through Gamification

Efan Eoin Haynes

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science by advanced study in Computer Science (conversion) in the Faculty of Engineering

---

September 2021 — CSMSC-21

Executive Summary

The concept of an autonomous vehicle (AV) has existed in literature for almost as long as cars themselves. With the technology of today we are closer than ever to making what was once fiction a reality. However, for an AV to be considered safe and road ready, testing must be done on a massive scale, potentially in the region of millions of miles. To avoid the dangers and costs of doing such testing on physical roads, eyes have turned to simulation. Here, we explore a method of creating challenging simulated driving scenarios aimed at contributing towards the confidence we have in the developing world of autonomous driving.

This paper researches the OpenScenario file format and how it may be used to describe complex driving scenarios when given only basic information. We will determine the limitations of the standard, and whether or not it is able to convert primitive data into simulated scenarios capable of verifying the driving logic of autonomous vehicles. The main deliverable of the project will be a developed converter that transforms logged data from any source into an OpenScenario file that will be capable of running on any supported simulator. Once complete, we then look to the suitability of the of the tools used and developed in achieving an additional method of verification.

Here is a list of the project contributions:

- Research done to identify whether the OpenScenario standard could be used as described above.
- A developed converter capable of adapting raw positional data into a universal driving standard file format.
- A developed analysis tool capable of numerical inspection, and an exploration of the results achieved.
- Identification of an appropriate metric of success and analysis the performance of this converter.
- An attempted explanation of the results achieved, and suggestions of alternative projects that could build on the knowledge gained throughout my research and development process.

Acknowledgements

I would like to thank my project supervisor Dr. Kerstin Eder for her motivating ideas that led to my pursuit of this project. Not only this, but for the direction she consistently provided me with throughout the period. I would also like to thank PhD student Abanoub Ghobrial for his limitless support, whether it be for his valuable insight on several of my ideas, or priceless patience and knowledge when assisting me with technical issues. Additionally, I extend my gratitude towards the Bristol Robotics Lab for their development and support of the Robovan Super AI game which pave the way for this projects larger contribution.

Finally, I would like to thank my friends and family for their encouragement throughout, most notably my girlfriend Jackie without whom this project would've been extremely stressful.

Ethics

This project did not require ethical review, as determined by my supervisor Kerstin Eder.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Efan Haynes
10th September 2021

COVID-19 Statement

The unavailability of university facilities, particularly at the onset of the project, due to the COVID-19 pandemic somewhat affected its progress. It was more difficult to get appropriate technical support in setting up my home machine for project work, since I could not use a lab machine and the university was unable provide me with another laptop. This significantly delayed the beginning of the development portion of my project and increased the stress already felt due to other factors associated with the pandemic i.e lockdown.

Fortunately, outside of this initial hiccup, my project was largely unaffected by COVID-19 due to its remote nature and lack of need for user feedback.

Contents

List of Figures

1 Introduction

1.1 Motivation

Verification, in the context of this paper, is defined as a process that measures a system's correctness against the specifications established for it [25]. Testing, one of the most common techniques employed in verification, is used to show that the actual behaviour of a system matches what is intended [27]. These two aspects of system design, along with validation (VVT), account for a large portion of of development cost, often estimated to be as high as 50-60 percent [2].

Despite this, while there exist standard practises, for example white or black box testing, there is no approach universally recognised as optimal for every system. This makes sense, as systems differ massively in their construction and use. As a result, there is still ongoing research into new, more efficient and effective methods of verification, with what I will cover being just one of these.

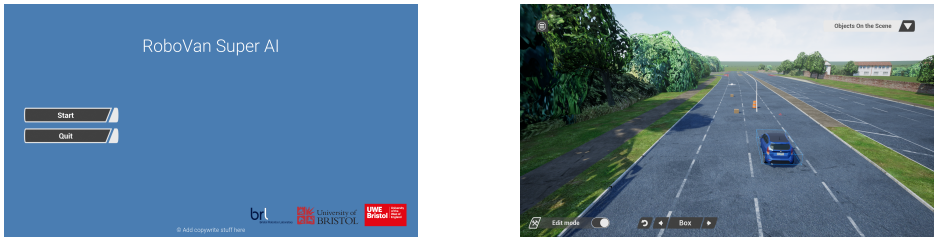
In the area we are concerned with, autonomous vehicles (AVs), it is not only the cost of verification that is an issue but also the safety. Simulation-based verification provides a fantastic tool for evaluating a test that would be too expensive or hazardous in practise. In the real world, vehicle testing is dangerous and thus would have to be carried out with extreme constraints. In a simulation, there are no such boundaries [4].

However, as it stands there is no clear way to select test stimuli or scenarios that lead to an *interesting* result. Here, interesting simply means that the information gathered increases the verification coverage, i.e we are confident in the behaviour of the vehicle in a wider selection of situations. Clearly, the verification coverage for a system governing the behaviour of an AV must be practically 100%, as the safety of other road users is paramount. Ideally, we would be able to generate a test for every single possibility in the simulation environment, but computationally this is not feasible due to the large parameter space of all complex simulators.

As such, research surrounding simulation-based verification in this context is primarily focused on finding the best way to gather those interesting tests. The intuitive method, requiring the least expertise, is randomising all the parameters and thus creating random scenarios. Unfortunately, an overwhelming majority of tests created in this manner are trivial and provide nothing of value. Contrarily, an expert manually creating scenarios is also a flawed approach, since this would take an inordinate amount of time.

My project attempts to meet these two opposing methods in the middle. Through the use of a pre-developed game, users will be able to easily create realistic scenarios where the vehicle must drive outside of normal circumstances. This could be achieved by the placement of static actors or other vehicles inside the game environment. These game scenarios will produce logs, whereupon a conversion tool I've developed will export these to a file type that more complex

simulators can interpret. The idea is that the game will work as an effective 'crowd-sourcing' of interesting scenarios without the technical expertise needed to understand CARLA or any other simulator. While I understand the scenarios created will be limited in scope by the constraints of the game, this project will serve as a proof of concept for the development of tests through gamification.



(a) Title screen (b) Creation of a scenario

Figure 1: Screenshots from the pre-developed game: Robovan SuperAI

1.2 Aims and Objectives

1.2.1 Aims

Now that I have sufficiently introduced the project area and necessary software knowledge will be shortly provided, I can concisely define this project's direction by these 3 aims:

1. Research the OpenScenario file type to determine whether it is possible to convert the data in the logs to a complete OpenScenario file, capable of re-creating the logged simulation. If yes, determine the limitations of the format.

2. Create a conversion tool that, when given this raw data about agents in a scenario, is able to convert this into an OpenScenario file that can be re-created across various simulators. Explore the accuracy of this converter, especially when a collision is involved.

3. After analysis, attempt to improve the performance of the converter, and then draw conclusions about the efficacy of this conversion method.

1.2.2 Objectives

To accomplish these, I set out key objectives that I consider milestones for the project's success.

1. Attempt to wholly understand the OpenScenario schema, and identify the best method to create the converter.

2. Begin development of the conversion tool by fetching the correct parameters to begin the scenario and integrating these into an automated Init section producer.

3. Continue development of the tool by deciding on the best way to convert the game logs to an OpenScenario story, and implementing this.

4. Test the accuracy of my converter by simulating several scenarios on simulators such as CARLA and esmini.

5. Formally analyse the difference between the logged simulation and my re-creation, attempting to identify any causes of variation and consider these.

2 Background and Context

2.1 Related Work

As previously mentioned, there have been studies on different methods of scenario generation, with the first I will go through based on an Agency-Directed Approach [10]. Carried out by people I am lucky enough to have supervising this project, here each agent is given a goal, and their behaviour is rewarded/punished with a score depending on how often/precisely these goals are met. The aim is to have the agents mimic realistic human behaviour by altering the goals to mirror human motivation in these scenarios, so that the agents become in a way intelligent, and act as a person would under the same circumstances. This results in realistic scenario generation far superior to pure random generation.

Another exciting avenue is the training of a Convolutional Neural Network(CNN) to generate realistic scenarios based on a set of a training data [9]. This Deep Learning algorithm, when trained on other expertly verified simulated scenarios, is capable of randomly generating a scenario map, and then based on the agents generated, creating a realistic scenario where each agent behaves in an expected manner. This method has been shown to achieve accuracy as high as 92%, and machine learning is clearly a very promising direction to go in terms of training autonomous vehicles. The only caveat is that the CNN was trained on simulated data, and so can only mimic human behavior as accurately as the simulator that generated said training data.

Finally, we look at the development of scenarios using crash data from existing automated components in cars [8]. While AVs are not used in full as of yet, features that are deemed safe are increasingly being allowed on the road, with the UK set to allow partially automated cars by the end of 2021 [16]. For a formal categorization of the levels of automation in an AV, please see the figure below. As it stands, most modern vehicles come fitted with Level 2 automated features, with companies in a race to develop level 3 aspects. Through analyzing crash data from the existing features, we are far more likely to develop scenarios where errors or improvements can be found.

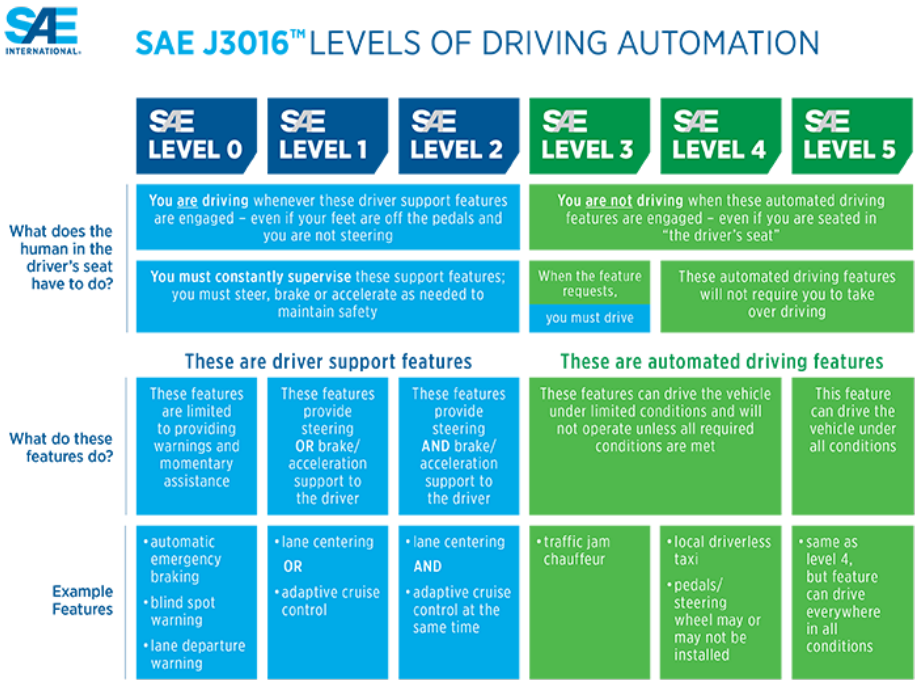


Figure 2: The SAE levels of driving automation

The difficulties facing the verification of autonomous vehicles are spelled out in Phillip Koopmans paper [6]. He claims that for autonomous vehicle failure rate to be the same as that of aircraft, a billion hours of testing must be carried out. It is up for debate as to whether this is a necessary level of verification, since the accident rate of manually driven cars is much higher than aircraft, but puts into perspective the sheer scale of safety precautions that must be taken. The paper then claims that given the hours required, system testing is not enough on its own. He suggests several alternative approaches, from the already discussed machine learning to phased deployment and fault injection.

Phased deployment follows the V model as a systematic process of verification and validation, adapted to autonomous vehicles as shown:

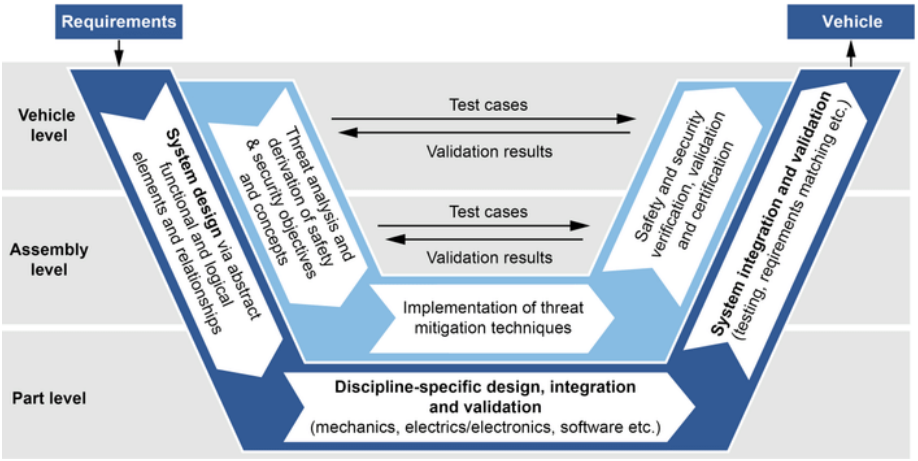


Figure 3: The V model for autonomous vehicle verification

Based on this model, I believe simulation can play a key part in the system integration and validation through large scale testing and immediate feedback.

There is a wealth of related work in this area, as it is in the public and private interest to successfully produce the first autonomous vehicle. However, almost every single method requires expert knowledge to develop and subsequently test these scenarios. This is the key area where I hope my project can contribute, in the sheer number of scenarios generated, since the method of scenario creation is simple.

2.2 Terminology and Key Concepts

2.2.1 Scenarios

Throughout this section I frequently refer to *scenarios*. Traditionally in language, a scenario is defined as 'A postulated sequence or development of events.' [17]. This captures the general meaning of a driving scenario rather well, however to avoid confusion a more formal description is required, including definitions of its constituent parts. These are not standardised definitions, but my interpretation, inspired predominantly by [7] and [5].

Definition 1 (Scenario)**.** A scenario describes the development of a sequence of traffic scenes in relation to time. Each scene comprises of a static environment, and various entities or agents undergoing actions. The movement of entities and agents throughout these scenes is characterised by Events. A scenario requires at least one initial scene, and ends as soon as a scene which is irrelevant to the scenario is reached.

Definition 2 (Scene)**.** A snapshot of the environments state, including scenery, dynamic elements and the relationships between these. Note that the snapshot does not specify a fixed time-frame.

Definition 3 (Static Environment)**.** The immovable features of a simulation, i.e weather, buildings.

Definition 4 (Entity)**.** Something which is part of the simulation but has no agency.

Definition 5 (Agent)**.** A part of the simulation that can execute Events based on the scenario state, i.e a pedestrian which can 'choose' when to cross the road.

Definition 6 (Action)**.** A basic change of parameter i.e Increasing the speed of a vehicle.

Definition 7 (Event)**.** Multiple Actions which combine to form an Event, i.e Actions slow down, indicate and change destination co-ordinate together form a left turn.

Efforts have been made in transferring this definition of a scenario into a defined XML framework, resulting in the release of OpenScenario.

3 Research

3.1 Standards

3.1.1 OpenDRIVE

For a scenario to take place, there must be a setting where the proposed actions can occur (Formally the 'Static Environment'). For the vast majority of real driving scenarios, this is some kind of road network, whether it be a roundabout, intersection or even a straight line. For all these, the basic 'rules of the road' [13] stay largely the same, as do the performance of vehicles. As such, it is natural for the first stage of AV simulation to involve creating a standard that mimics these road networks.

With such a standard in place, any simulated AV will be restricted in how it drives, much like we are in how we drive along roads. This includes necessities such as correct lane assignment, but also can communicate to the simulator different road conditions (e.g crossfall [24], elevation, surface quality) that would affect how a vehicle controls.

Thus, in 2005, work began on this by companies Daimler Driving Simulator, Stuttgart and VIRES Simulationstechnologie GmbH, under the name OpenDRIVE . When the initiative was publicized on 2006, several others became interested. OpenDrive is still the go-to standard today, now under the name ASAM OpenDRIVE since the transfer to ASAM in 2018.

An OpenDRIVE(.xodr) file is of an XML format with a hierarchical structure, and provides the following information: [28]

- Analytical definition of road geometry
- Plane elements, elevation, crossfall, lane width etc.
- Various types of lanes
- Logical inter-connection of lanes
- Signal controllers (e.g. for junctions)
- Road surface properties
- Road and road-side objects

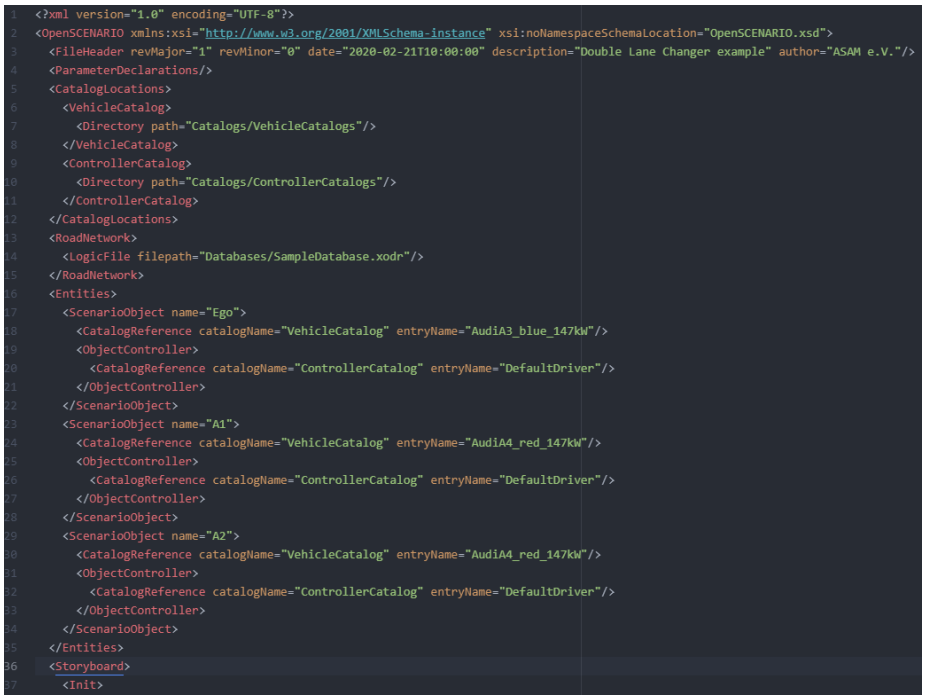
These combined completely describe the way a vehicle may drive around the road network.

3.1.2 OpenScenario

Now that the road network has been described to the simulator, all that remains is to do the same for the dynamic content of the scenario, in a similar fashion as OpenDRIVE. Before industry standards like OpenScenario, the description of this dynamic content was done by hand and the implementation varied from simulator to simulator. This was the status quo for several years, before development began on OpenScenario.

In January of 2020, ASAM OpenScenario 1.0.0 was released, and while previous versions exist, this was the pivotal point where most simulators began implementing OpenScenario support. By support, I mean an OpenScenario file is input, converted, and then compiled as Simulator code, although not all features of OpenScenario are supported by every simulator. Much like its sister OpenDRIVE, an OpenScenario(.xosc) file is of an XML format with a hierarchical structure, and attempts to capture the definition of a scenario inside the XML Schema.

There are 3 sections to an OpenScenario file, topline definitions, Init, and the Stories. In the topline definitions, the simulator, road network, and entities/-controllers are declared, along with basic file details. A controller details how an entity is handled by the simulator. In the Init section, the initial positions and speeds of entities are set. Here are some examples of these two sections, taken from the ASAM example group.



```
<?xml version="1.0" encoding="UTF-8"?>
<OpenSCENARIO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="OpenSCENARIO.xsd">
  <FileHeader revMajor="1" revMinor="0" date="2020-02-21T10:00:00" description="Double Lane Changer example" author="ASAM e.V."/>
  <ParameterDeclarations/>
  <CatalogLocations>
    <VehicleCatalog>
      <Directory path="Catalogs/VehicleCatalogs"/>
    </VehicleCatalog>
    <ControllerCatalog>
      <Directory path="Catalogs/ControllerCatalogs"/>
    </ControllerCatalog>
  </CatalogLocations>
  <RoadNetwork>
    <LogicFile filepath="Databases/SampleDatabase.xodr"/>
  </RoadNetwork>
  <Entities>
    <ScenarioObject name="Ego">
      <CatalogReference catalogName="VehicleCatalog" entryName="AudiA3_blue_147kW"/>
      <ObjectController>
        <CatalogReference catalogName="ControllerCatalog" entryName="DefaultDriver"/>
      </ObjectController>
    </ScenarioObject>
    <ScenarioObject name="A1">
      <CatalogReference catalogName="VehicleCatalog" entryName="AudiA4_red_147kW"/>
      <ObjectController>
        <CatalogReference catalogName="ControllerCatalog" entryName="DefaultDriver"/>
      </ObjectController>
    </ScenarioObject>
    <ScenarioObject name="A2">
      <CatalogReference catalogName="VehicleCatalog" entryName="AudiA4_red_147kW"/>
      <ObjectController>
        <CatalogReference catalogName="ControllerCatalog" entryName="DefaultDriver"/>
      </ObjectController>
    </ScenarioObject>
  </Entities>
  <Storyboard>
    <Init>
```

(a) Topline Content

```
<Init>
  <Actions>
    <Private entityRef="Ego">
      <PrivateAction>
        <LongitudinalAction>
          <SpeedAction>
            <SpeedActionDynamics dynamicsShape="step" value="0" dynamicsDimension="time"/>
            <SpeedActionTarget>
              <AbsoluteTargetSpeed value="3.6111111111111107e+01"/>
            </SpeedActionTarget>
          </SpeedAction>
        </LongitudinalAction>
      </PrivateAction>
      <PrivateAction>
        <TeleportAction>
          <Position>
            <WorldPosition x="1.7024039832784507e+02" y="3.4330477905273438e+02" z="0.0000000000000000e+00" h="1.5707963267948966e+00" p="0.0000000000000000e+00" r="0.0000000000000000e+00"/>
          </Position>
        </TeleportAction>
      </PrivateAction>
    </Private>
    <Private entityRef="A1">
      <PrivateAction>
        <LongitudinalAction>
          <SpeedAction>
            <SpeedActionDynamics dynamicsShape="step" value="0" dynamicsDimension="time"/>
            <SpeedActionTarget>
              <AbsoluteTargetSpeed value="4.7222222222222221e+01"/>
            </SpeedActionTarget>
          </SpeedAction>
        </LongitudinalAction>
      </PrivateAction>
      <PrivateAction>
        <TeleportAction>
          <Position>
            <WorldPosition x="1.6682571411132812e+02" y="3.3006811523437500e+02" z="0.0000000000000000e+00" h="1.5707963267948966e+00" p="0.0000000000000000e+00" r="0.0000000000000000e+00"/>
          </Position>
        </TeleportAction>
      </PrivateAction>
    </Private>
    <Private entityRef="A2">
      <PrivateAction>
        <LongitudinalAction>
          <SpeedAction>
            <SpeedActionDynamics dynamicsShape="step" value="0" dynamicsDimension="time"/>
            <SpeedActionTarget>
              <AbsoluteTargetSpeed value="3.6111111111111107e+01"/>
            </SpeedActionTarget>
          </SpeedAction>
        </LongitudinalAction>
      </PrivateAction>
      <PrivateAction>
        <TeleportAction>
          <Position>
            <WorldPosition x="1.7024039832784504e+02" y="4.1330477905273438e+02" z="0.0000000000000000e+00" h="1.5707963267948966e+00" p="0.0000000000000000e+00" r="0.0000000000000000e+00"/>
          </Position>
        </TeleportAction>
      </PrivateAction>
    </Private>
  </Actions>
</Init>
```

(b) Init Section

Figure 4: The initial two sections of an OpenScenario file, defining the initial 'scene'.

There are several intricacies in correctly formatting these sections, and extras such as the use of catalogues, but these are not of import to the project, so are not discussed. Full documentation can be found here by downloading their 'User guide' [12].

The final section, a collection of 'Stories' formally describe the interaction between entities and the world in a scenario. The XML classes we are interested in show their relation in this UML Class Diagram.

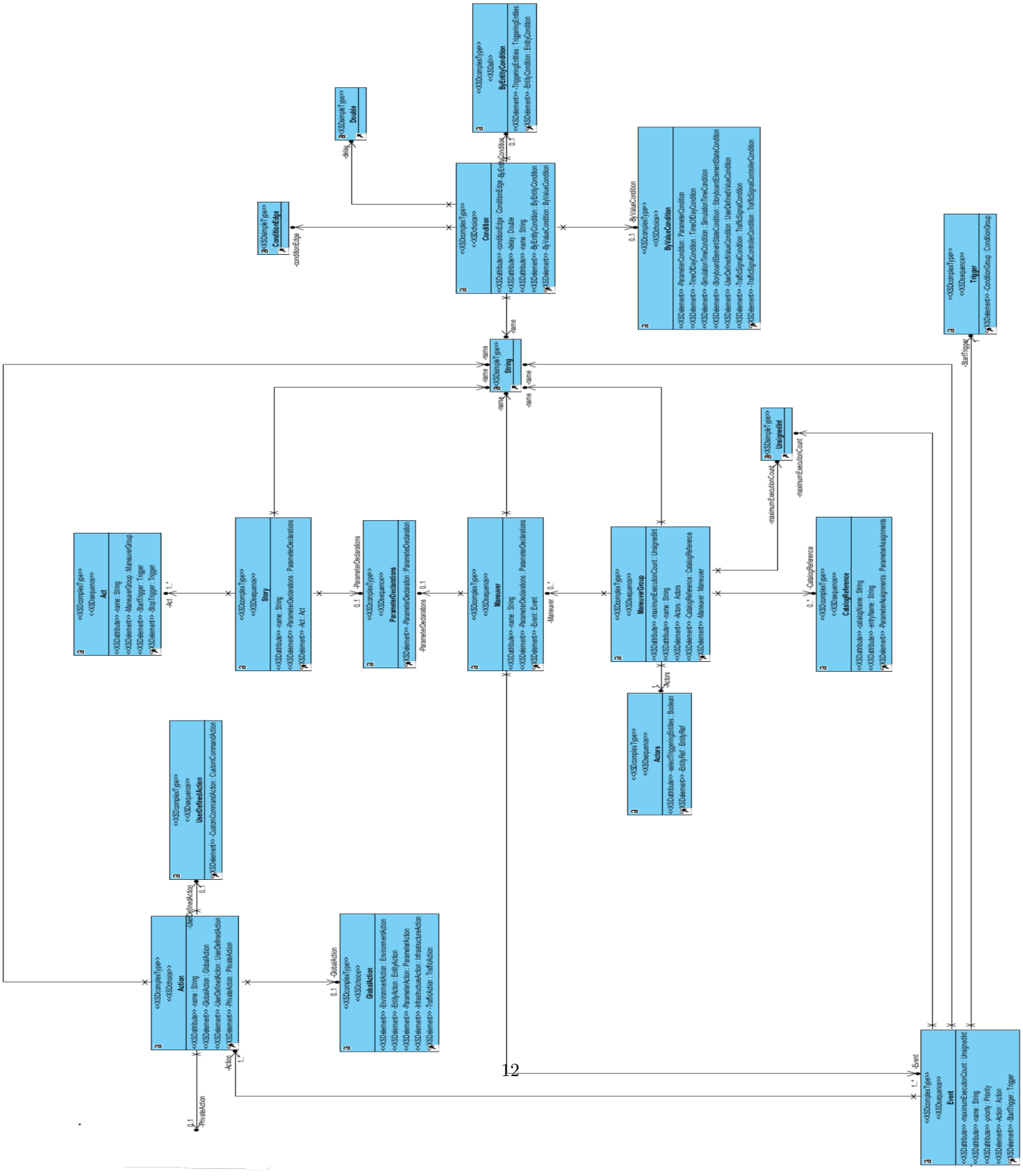


Figure 5: A UML Diagram showing the relation between the essential OpenScenario classes.

Although initially daunting, the main relations to note here are between:

- <Maneuver> → <Event>
- <Event> → <Action>
- <Maneuver> → <Trigger>
- <Event> → <Trigger>

In essence, a Maneuver (for example an overtake) is broken down in several Events, perhaps, a 'SpeedUp event' and 'Change lane event'x2. Each Event is further abstracted into actions, e.g the SpeedUp event includes an <AbsoluteTargetSpeed> action with various parameters. These events begin when their condition is triggered, in our example when the two cars involved reach a certain relative distance.

Here is an example of the beginning of a story in OpenScenario:

```
<Story name="MyStory">
  <ParameterDeclarations>
    <ParameterDeclaration parameterType="string" name="$owner" value="A1"/>
  </ParameterDeclarations>
  <Act name="MyAct">
    <ManeuverGroup maximumExecutionCount="1" name="MySequence">
      <Actors selectTriggeringEntities="false">
        <EntityRef entityRef="$owner"/>
      </Actors>
      <Maneuver name="MyDoubleLaneChangeManeuver">
        <Event name="MyLaneChangeRightEvent" priority="overwrite">
          <Action name="MyLaneChangeRightAction">
            <PrivateAction>
              <LateralAction>
                <LaneChangeAction>
                  <LaneChangeActionDynamics dynamicsShape="sinusoidal" value="2" dynamicsDimension="time"/>
                  <LaneChangeTarget>
                    <RelativeTargetLane entityRef="$owner" value="-1"/>
                  </LaneChangeTarget>
                </LaneChangeAction>
              </LateralAction>
            </PrivateAction>
          </Action>
          <StartTrigger>
            <ConditionGroup>
              <Condition name="MyStartCondition1" delay="0" conditionEdge="rising">
                <ByEntityCondition>
                  <TriggeringEntities triggeringEntitiesRule="any">
                    <EntityRef entityRef="$owner"/>
                  </TriggeringEntities>
                  <EntityCondition>
                    <DistanceCondition value="5.0000000000000000e+00" freespace="false" alongRoute="false" rule="greaterThan">
                      <Position>
                        <RelativeObjectPosition entityRef="Ego" dx="0.0000000000000000e+00" dy="0.0000000000000000e+00"/>
                      </Position>
                    </DistanceCondition>
                  </EntityCondition>
                </ByEntityCondition>
              </Condition>
            </ConditionGroup>
          </StartTrigger>
        </Event>
        <Event name="MyLaneChangeLeftEvent" priority="overwrite">
          <Action name="MyLaneChangeLeftAction">
            <PrivateAction>
              <LateralAction>
                <LaneChangeAction>
                  <LaneChangeActionDynamics dynamicsShape="sinusoidal" value="2" dynamicsDimension="time"/>
                  <LaneChangeTarget>
                    <RelativeTargetLane entityRef="$owner" value="1"/>
                  </LaneChangeTarget>
                </LaneChangeAction>
              </LateralAction>
            </PrivateAction>
          </Action>
          <StartTrigger>
            <ConditionGroup>
              <Condition name="MyStartCondition2" delay="0" conditionEdge="rising">
                <ByValueCondition>
                  <StoryboardElementStateCondition storyboardElementType="action" storyboardElementRef="MyLaneChangeRightAction" state="completeState"/>
                </ByValueCondition>
```

OpenScenario is an incredible standard, and it is possible to create very complex scenarios with few lines of code. My project does not use the full functionality due to limitations of the project scope and simulator used. I would highly recommend to further read and understand the OpenScenario standard if this work interests you.

3.2 OpenScenario Project Relevance

The first step in any development task is to understand the problem you're trying to solve, and then evaluate whether the tools at your disposal are adequate for the task. As such, my first foray into the world of AVs was aimed at understanding the OpenScenario file format in general, much like how I explained it above. Given the plethora of resources available to aid the theoretical understanding and examples to confirm it, this was not too challenging. The true difficulty was how to take this format, where files are usually created with a specific scenario in mind, and adapt it for use in a general situation.

The largest issue was that something this has never been done so generally before, so there was little reference or guidance as to whether this was a feasible idea. Nonetheless, here is what I found.

3.2.1 Intended use

While I have explored the basics of what OpenScenario is, and some ways in which it works, I have yet to discern what was in mind when it was designed, and whether this suits the project goals. According once again to the ASAM User guide, 'OpenSCENARIO is used in virtual development, test, and validation of functions for driver assistance, automated driving and autonomous driving' [12]. This aligns almost exactly with my project's goals, since we also focus on verification and validation.

To further confirm the possibility of this project, I looked to see whether there are any other niche automated parsers or converters already present. These need not look to generate scenarios as ours will, perhaps simply take an already generated scenario and parse it into the OpenScenario format. In doing so, I found the MathWorks 'Automated Driving Toolbox', with full documentation found online [20]. Although I could not learn of its capabilities given that it is a paid service, it boasts a large collection of features from AV simulation to Lidar processing [22].

The feature I was interested in was the option to 'export to OpenScenario'. This gives the user an the option to convert a 'driving scenario' into an OpenScenario file format. The driving scenario in question must either be one explicitly defined in the conversion process, or have been previously created by MatLab's *drivingScenarioDesigner*. While this is less general than what we hope to achieve, it shows that some kind of conversion is indeed possible.

Please find here the process of converting a scenario to OpenScenario in this manner, and the data required: [21]

Create a driving scenario.

```
scenario = drivingScenario('StopTime',6);
```

Import the road network from an OpenDRIVE file into the scenario.

```
fileName = 'parking.xodr';
roadNetwork(scenario,'OpenDRIVE',fileName);
```

Add an ego vehicle to the scenario. Set a trajectory in which the vehicle drives along the curve at varying speed.

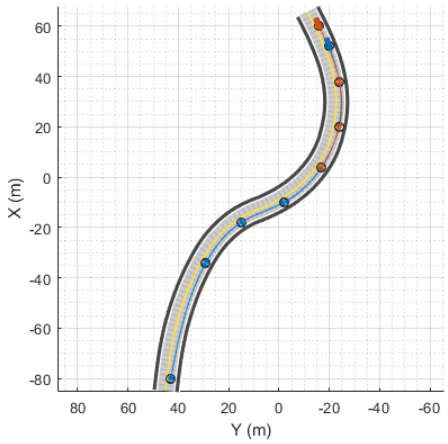
```
egoVehicle = vehicle(scenario,'ClassID',1);
waypoints = [-80 43; -34 29; -18 15; -10 -2; 4 -17; 38 -24; 52 -20];
speed = [50 20 20 20 20 50 50];
trajectory(egoVehicle,waypoints,speed);
```

Add a non-ego actor and set it to spawn during the simulation by specifying an entry time value. Generate a trajectory for the non-ego actor.

```
truck = vehicle(scenario,'ClassID',2,'Position',[4 -17 0],'EntryTime',3);
waypoints = [4 -17; 20 -24; 38 -24; 60 -16];
speed = [40 40 40 40];
trajectory(truck,waypoints,speed);
```

Plot the scenario and run the simulation. Observe how the vehicle slows down as it drives along the curve.

```
plot(scenario,'Waypoints','on');
while advance(scenario)
    pause(0.01)
end
```



Export the scenario to an ASAM OpenSCENARIO file.

```
export(scenario,'OpenSCENARIO','parking.xosc');
```

Figure 6: Creation of a convertible scenario

The existence of this tool leads me to believe that the hypothesised conversion

tool is possible, as much of the automated OpenScenario content would be the same. However, there is still one key difference. Here, the speed, waypoints, trajectory and even stopping time are already defined in the scenario, giving a clear picture of how the scenario should look. In what I aim to build, there would not be access to such high level information. From this we can narrow down the challenge to how to best express an entitiy's path and when it should end with the limited data we are given.

We are also unable to corroborate whether or not the produced OpenScenario files perform as expected on any simulator outside of the one provided with MathWorks's driving toolbox, the ability to test this would be very useful. Neverthesless, the recommended direction for OpenScenario's use, along with the existence of a somewhat similar tool give me confidence in successful development of the proposed converter.

4 Development

4.1 Introduction

What was found in the above section led me to believe that an accurate converter could be created, granted that entity paths have the potential to be calculated. My approach in doing so, and the steps I took to develop my converter are outlined in the following sections.

4.2 Tools

The two primary tools used in the development of my converter were Java (JDK 11) for the code, and CARLA (powered by Unreal Engine 4) to re-create the scenarios generated. Traditionally, the go-to development language relating to CARLA is python, considering if you are to interact with a CARLA simulation it is normally done through the python API. However, since the converter will stand completely seperate to CARLA and work across platforms, I was free to use Java as I have more experience.

While several companies exist whose simulators support OpenScenario (Cognata, dSpace etc.). The vast majority of these are not free. Of the free, open source, high level few, (SVL, OpenDS, etc.) CARLA has by far the most OpenScenario support. Because of this, CARLA was the natural choice.

4.3 CARLA Limitations

4.3.1 Trajectories

Despite being the most well supported, there are still several key features in OpenScenario that cannot be processed by CARLA. A full guide on what is possible can be found on the CARLA website [14]. The most impactful omission is the missing 'FollowTrajectoryAction'. If this were included, it would be a simple matter of defining the path for each vehicle to take wholly described by a trajectory. This could either be a Polyline, Clothoid or groupings of Non-Uniform Rational B-Splines (Nurbs) of any order (3 or 4 being most common). Since this is not available, we must resort to far more basic ideas when setting the entity path.

4.3.2 Non-determinism

Something else to consider when working with CARLA or game engines in general is their non-deterministic nature. This means that if a scenario is created with the same actors, entities and environment, all with the exact same history, the resulting simulations may not be consistent. This is clearly a large issue when attempting to verify autonomous vehicles, as the correct behaviour is not guaranteed, leading some to question to capability of game engines as simulators full stop. In order to tackle this, efforts to identify the causes of this variance

and reduce them have been made [11]. The key aspect found was an idea of *tolerance*, or an acceptable level of variance in a scenario. The effective ways to reduce this variance are to avoid collisions and monitor the utilisation of system resources during execution. Since both the scenario generation and execution will be in the hands of the converters users, while I am aware of these issues, I cannot stop them, merely notify the users.

Work has already begun on deterministic simulators, and these will be compatible with OpenScenario, so eventually this converter will be able to be used on a deterministic platform.

4.4 Development Approach - CARLA

To achieve our first aim, and succesfully convert from the game logs to an OpenScenario file, we must first analyse the logs produced. Here is a sample:

```
testNo, repeatNo, agentNo, agentID, agentType, agentTypeNo, x, y, z, yaw, vel_x, vel_y, vel_z, speed, time, sim_time, fps
0, 0, 1, 1679, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.545, -2.094, 0.000, 0.000, -2.935, 2.935, 0.000, 978.104, 0.000
0, 0, 2, 1680, walker.pedestrian.0002, 1, -22.240, -62.490, 1.694, 0.000, 0.000, 0.000, -2.450, 2.450, 0.000, 978.104, 0.000
0, 0, 1, 1679, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.198, -2.094, 0.000, 0.000, -3.912, 3.912, 0.100, 978.204, 0.000
0, 0, 2, 1680, walker.pedestrian.0002, 1, -22.249, -62.565, 1.400, -0.509, 0.010, -0.005, -3.430, 3.430, 0.100, 978.204, 0.000
0, 0, 1, 1679, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 0.753, -2.094, 0.000, 0.000, -4.888, 4.888, 0.200, 978.304, 0.000
0, 0, 2, 1680, walker.pedestrian.0002, 1, -22.268, -62.714, 1.008, -1.116, 0.022, -0.024, -4.410, 4.410, 0.200, 978.304, 0.000
0, 0, 1, 1679, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 0.210, -2.094, 0.000, 0.000, -5.862, 5.862, 0.300, 978.404, 0.000
0, 0, 2, 1680, walker.pedestrian.0002, 1, -22.285, -62.873, 0.957, -1.413, 0.051, -0.226, 0.000, 0.232, 0.300, 978.404, 0.000
0, 0, 1, 1679, vehicle.mercedes-benz.coupe, 3, -26.261, -7.952, 0.062, -2.094, -0.002, 0.003, 0.211, 0.211, 0.400, 978.504, 0.000
0, 0, 2, 1680, walker.pedestrian.0002, 1, -22.305, -63.033, 0.957, -1.558, 0.030, -0.451, 0.000, 0.452, 0.400, 978.504, 0.000
```

Figure 7: Logs produced by a run of the game

Not all of the of the information present here is necessary, and the velocity values are currently not accurate, so these will not be used. Of the remaining columns, let us define their use, if any.

Log Feature	Use
testNo	N/A
repeatNo	N/A
agentNo	An entity reference in the converter
agentID	N/A
agentType	Spawn in the correct CARLA actors
agentTypeNo	N/A
x	X position in the CARLA world
y	Y position in the CARLA world
z	Z position in the CARLA world
yaw	orientation of the actor in the CARLA world
time	Aid in velocity calculation during conversion

Figure 8: A table showing the uses of each data column in the converter

The first step is to parse all of the required data into a java class. I chose to do this individually for each line of the logs, and then have these added to an arraylist. This was a rather simple affair, and involved basic file reading and string splitting. Here a snapshot of the code:

```
private ArrayList<logLine> getLines (File fileToRead) {
    String lineToRead;
    ArrayList<logLine> lines = new ArrayList<>();
    try {
        if (fileToRead.length() != 0) {
            fileReader = new FileReader(fileToRead);
            buffReader = new BufferedReader(fileReader);
            buffReader.readLine();
            while ((lineToRead = buffReader.readLine()) != null) {
                lines.add(parseLine(lineToRead));
            }
        }
    }
    catch (FileNotFoundException e) {
        System.out.println("Specified file does not exist!");
    }
    catch (IOException e) {
        System.out.println("Could not read from the file!");
    }
    return lines;
}

private logLine parseLine (String lineToParse) {
    logLine line;
    String[] splitString;
    splitString = lineToParse.split( regex: ",");
    for (int i = 0; i < splitString.length; i++) {
        splitString[i] = splitString[i].trim(); //Remove whitespace
    }
    //Here I insert the necessary quantities from the logs into a separate Class for later use
    line = new logLine(splitString);
    return line;
}
```

Figure 9: Java methods to parse in data

```java
public class logLine {
    //Class that holds the relevant values from a line of the produced log file
    private int agentNo;
    private String agentType;
    private double x;
    private double y;
    private double z;
    private double yaw;
    private double simTime;

    public logLine (String[] splitString) {
        agentNo = Integer.parseInt(splitString[2]);
        agentType = splitString[4];
        x = Double.parseDouble(splitString[6]);
        y = Double.parseDouble(splitString[7]);
        z = Double.parseDouble(splitString[8]);
        yaw = Double.parseDouble(splitString[9]);
        simTime = Double.parseDouble(splitString[14]);
    }

    public int getAgentNo () { return agentNo; }

    public String getAgentType () { return agentType; }

    public double getX () { return x; }

    public double getY () { return y; }
```

Figure 10: Java class to store the logged data

4.4.1 First steps

Now that we have all the information, the problem became how best to express this data in OpenScenario. The topline section of the file does not change much between scenarios so this could be hard-coded with explicit strings. The only additions of note here in comparison to the example scenario was including CARLA: in the description to ensure the correct co-ordinate system and introducing basic functions to pull the date/time and map from the command line.

4.4.2 Init

Next, we look to tackling the init section, or re-creating the first 'scene' of the scenario given. The primary task was to successfully initialise all the entities with the correct properties. This was done by looking through the first log entry for each entity (deducible by the largest agentNo) and taking the associated agentType. If the agentType matches a known CARLA vehicle, this is spawned in, else a default decided by the entity category is chosen [14]. Assuming the vehicle is not a CARLA standard, the category must be specified in the first word of the agentType, as this is only place in the logs where it can be done. Since different categories require different properties, this is essential, e.g the

Vehicle class does not need the 'Mass' parameter, whereas a pedestrian does. Despite several of these required parameters remaining unused by CARLA, their presence is needed to parse the XML schema without error. The final piece of the init section is the use of a <Teleport> action, initialising the referenced vehicles at the correct world position at time 0 in the logs. Here are some snapshots of code from this section:

NOTE: Some trivial sections are skipped, replaced with ellipses.

```
private void writeInEntities () {
       try {
           fileWriter = new FileWriter(fileToCreate, true);
           buffWriter = new BufferedWriter((fileWriter));
           buffWriter.write(" <Entities>");
           buffWriter.newLine();
           classifyObject();
           buffWriter.write(" </Entities>");
           buffWriter.newLine();
           buffWriter.flush();
           buffWriter.close();
       }
       catch (IOException e) {
           System.out.println("Could not write to the file!");
           System.exit(1);
       }
   }
```

```
private void initActions () {
       try {
           fileWriter = new FileWriter(fileToCreate, true);
           buffWriter = new BufferedWriter((fileWriter));
           buffWriter.write(" <Storyboard>");
           buffWriter.newLine();
           buffWriter.write(" <Init>");
           buffWriter.newLine();
           buffWriter.write((" <Actions>"));
           buffWriter.newLine();
           for (int i = 0; i < parseInput.getEntityNo(); i++) {
               buffWriter.write("  <Private entityRef=\"" + (i+1) +
                   "\">");
               buffWriter.newLine();
               buffWriter.write("   <PrivateAction>");
               buffWriter.newLine();
               buffWriter.write("    <TeleportAction>");
               buffWriter.newLine();
               buffWriter.write("     <Position>");
               buffWriter.newLine();
               buffWriter.write("      <WorldPosition x=\"" +
                   logLines.get(i).getX() + "\" y=\"" +
                   logLines.get(i).getY() +
                      "\" z=\"" + logLines.get(i).getZ() + "\" h=\"" +
                          logLines.get(i).getYaw() + "\"/>");
               buffWriter.newLine();
               buffWriter.write("     </Position>");
               ...
               ...
           buffWriter.write(" </Init>");
           buffWriter.newLine();
           buffWriter.flush();
           buffWriter.close();
       }
       catch (IOException e) {
           System.out.println("Could not write to the file!");
           System.exit(1);
       }
   }
```

```
private void classifyObject () {
       try {
           for (int i = 0; i < parseInput.getEntityNo(); i++) {
               buffWriter.write(" <ScenarioObject name=\"" + (i+1) +
                   "\">");
               buffWriter.newLine();
               if (logLines.get(i).getAgentType().charAt(0) == 'v') {
                   buffWriter.write(" <Vehicle name=\"" +
```

```
                    logLines.get(i).getAgentType() + "\"
                    vehicleCategory=\"car\">");
                buffWriter.newLine();
                buffWriter.write("  <ParameterDeclarations/>");
                buffWriter.newLine();
                buffWriter.write("  <Performance maxSpeed=\"69.444\"
                    maxAcceleration=\"200\"
                    maxDeceleration=\"10.0\"/>");
                ...
                ...
                ...
                buffWriter.write("   <Property name=\"type\"
                    value=\"pedestrian" + i + "\"/>");
                buffWriter.newLine();
                buffWriter.write("  </Properties>");
                buffWriter.newLine();
                buffWriter.write(" </Pedestrian>");
                buffWriter.newLine();
            }
            buffWriter.write(" </ScenarioObject>");
            buffWriter.newLine();
        }
    }
```

Figure 11: Code of the scenario's Init section

4.4.3 Story

Now we must decide on the best approach, given that trajectories cannot be used. Again, from the list of supported actions, the list of available RoutingActions are AcquirePositionAction and AssignRouteAction. Since AcquirePositionAction is limited to following pre-defined splines based on the map content, this cannot be used as not all scenarios would be accurately reflected. As such, we must create and then assign a route for each entity to follow. In addition to this, the speed of the vehicle must somehow be reflected. I chose to calculate the average velocity of each entity between each pair of waypoints, and have them sustain this speed throughout the traversal period. Changing the speed based on proximity at more frequent intervals can lead to *oversampling*, fully explored later. Utilising average speed will have a negligible effect on the accuracy of the simulation, given relatively short spacing between two waypoints. However, things change if a collision takes place, since a slight change in velocity during a collision can have a large impact on the physics of the crash.

Another approach was considered, rather than updating the speed at each waypoint, instead do so at fixed time intervals. This can be done much more frequently than proximity-based due to the size of the entity not being a factor.

However, despite looking promising, this was actually less reliable during collisions, since one slight difference in contact (common due to non-determinism), would send the entire scenario out of sync. This was deemed worse since a hiccup at the beginning of the scenario would cause incorrect speed calculations for the rest of the simulation.

The story code takes the values from the logs and creates a separate ManeuverGroup for each vehicle, since the entity referenced per ManeuverGroup is unique. Inside each Group is a single maneuver named 'FollowWaypointsX' where X is the entity counter, beginning at 0. This maneuver consists of an unknown quantity (depends on the logs given) of events, one which sets the route by calculation of the waypoints, the others which set the speed of the entity when it reaches each of these waypoints.

Here are some snippets of the most important pieces of code:

```
private void mapWayPoints(BufferedWriter buffWriter, int AgentNumber,
    int sourceNumber) {
      try {
          int entityNumber = parseInput.getEntityNo();
          wayPoints.add(logLines.get(AgentNumber-1));
          for (int i = AgentNumber-1; i < logLines.size()-1 -
              entityNumber; i = i + entityNumber) {
              if (calculateAverage(logLines.get(sourceNumber),
                  logLines.get(i), "Distance") >= lookahead) {
                  velocities.add(calculateAverage(logLines.get(sourceNumber),logLines.get(i),
                      "Speed"));
                  wayPoints.add(logLines.get(i));
                  sourceNumber = i;
                  buffWriter.write("         <Waypoint
                      routeStrategy=\"shortest\">");
                  buffWriter.newLine();
                  buffWriter.write("          <Position>");
                  buffWriter.newLine();
                  buffWriter.write("           <WorldPosition x=\"" +
                      logLines.get(i).getX() + "\" y=\"" +
                      logLines.get(i).getY() + "\" z=\"" +
                      logLines.get(i).getZ() + "\" h=\"" +
                      logLines.get(i).getYaw() + "\"/>");
                  buffWriter.newLine();
                  buffWriter.write("          </Position>");
                  buffWriter.newLine();
                  buffWriter.write("         </Waypoint>");
                  buffWriter.newLine();
              }
          }
          count++;
      }
      catch (IOException e) {
          System.out.println("Could not write waypoints to the file!");
```

```
    }
  }
```

```
private void setSpeedEvents(BufferedWriter buffWriter, int AgentNumber)
    {
      try {
          for (int i = 0; i < velocities.size(); i++, count++) {
              buffWriter.write("    <Event name =\"RouteEvent" + count +
                  "\" priority=\"parallel\">");
              buffWriter.newLine();
              buffWriter.write("     <Action name =\"ActionSpeed" +
                  count + "\">");
              buffWriter.newLine();
              buffWriter.write("      <PrivateAction>");
              buffWriter.newLine();
              buffWriter.write("       <LongitudinalAction>");
              buffWriter.newLine();
              buffWriter.write("        <SpeedAction>");
              buffWriter.newLine();
              buffWriter.write("         <SpeedActionDynamics
                  dynamicsShape=\"step\" value=\"0\"
                  dynamicsDimension=\"time\"/>");
              buffWriter.newLine();
              buffWriter.write("         <SpeedActionTarget>");
              buffWriter.newLine();
              buffWriter.write("          <AbsoluteTargetSpeed value=\""
                  + velocities.get(i) + "\"/>");
              buffWriter.newLine();
              ...
              ...
              buffWriter.write("     </Action>");

              buffWriter.newLine();
              buffWriter.write("     <StartTrigger>");
              buffWriter.newLine();
              buffWriter.write("      <ConditionGroup>");
              buffWriter.newLine();
              buffWriter.write("       <Condition name=\"\" delay=\"0\"
                  conditionEdge=\"none\">");
              buffWriter.newLine();
              buffWriter.write("        <ByEntityCondition>");
              buffWriter.newLine();
              buffWriter.write("         <TriggeringEntities
                  triggeringEntitiesRule=\"any\">");
              buffWriter.newLine();
              buffWriter.write("          <EntityRef entityRef=\"" +
                  AgentNumber + "\"/>");
              buffWriter.newLine();
              buffWriter.write("         </TriggeringEntities>");
```

```
                buffWriter.newLine();
                buffWriter.write("        <EntityCondition>");
                buffWriter.newLine();
                buffWriter.write("         <ReachPositionCondition
                    tolerance=\"2\">");
                buffWriter.newLine();
                buffWriter.write("          <Position>");
                buffWriter.newLine();
                buffWriter.write("           <WorldPosition x=\"" +
                    wayPoints.get(i).getX() + "\" y=\"" +
                    wayPoints.get(i).getY() + "\" z=\"" +
                    wayPoints.get(i).getZ() + "\"/>");
                buffWriter.newLine();
                buffWriter.write("          </Position>");
                ...
                ...
                buffWriter.write("    </Event>");
                buffWriter.newLine();
            }
            velocities.clear();
            wayPoints.clear();
        }
        catch (IOException e) {
            System.out.println("Cannot write velocities to the file!");
        }
    }
```

```
private void eventActions (BufferedWriter buffWriter, int AgentNumber) {
        try {
            int entityNumber = parseInput.getEntityNo();
            double avgSpeed;
            buffWriter.write("    <Event name =\"RouteEvent" + count +
                "\" priority=\"overwrite\">");
            buffWriter.newLine();
            buffWriter.write("     <Action name =\"Assign Route\">");
            buffWriter.newLine();
            buffWriter.write("      <PrivateAction>");
            buffWriter.newLine();
            buffWriter.write("       <RoutingAction>");
            buffWriter.newLine();
            buffWriter.write("        <AssignRouteAction>");
            buffWriter.newLine();
            buffWriter.write("         <Route name=\"Route 1\" closed =
                \"false\">");
            buffWriter.newLine();
            mapWayPoints(buffWriter, AgentNumber, AgentNumber - 1);
            buffWriter.write("         </Route>");
            buffWriter.newLine();
            buffWriter.write("        </AssignRouteAction>");
```

```
            buffWriter.newLine();
            buffWriter.write("      </RoutingAction>");
            buffWriter.newLine();
            buffWriter.write("     </PrivateAction>");
            buffWriter.newLine();
            buffWriter.write("    </Action>");
            buffWriter.newLine();
            buffWriter.write("    <StartTrigger>");
            ...
            ...
            buffWriter.newLine();
            buffWriter.write("    </StartTrigger>");
            buffWriter.newLine();
            buffWriter.write("   </Event>");
            buffWriter.newLine();
            setSpeedEvents(buffWriter, AgentNumber);
        }
        catch (IOException e) {
            System.out.println("Could not write to the file!");
            System.exit(1);
        }
    }
```

Before delving in to the calculation of the waypoints, I will note that the StartTrigger for the story is simulation time ≥ 0 seconds. The scenario automatically ends when all events are complete, or the StopTrigger is activated at simulation time ≥ 30 seconds (easily changeable).

4.5 Waypoints & Lookahead distance

The spacing of waypoints for an autonomous vehicle to follow is known as the *lookahead distance*, i.e how far an entity looks forward for its next goal. It is an important feature of any route, and bad implementation can lead to entities following the path incorrectly. For example, consider a vehicle travelling at a relatively high velocity, turning onto a new stretch of road. If the vehicle overshoots the intended path even slightly when turning, very small lookahead distances would cause the vehicle to oscillate along the path [1]. This happens due to the vehicle driving at this high speed at an angle to the road, consistently overshooting the frequent waypoints. Too sparse an implementation of waypoints also causes problems, oftentimes causing the vehicle to completely miss bends in favour of a more direct route.

These issues are more intuitively shown in the figure below [23].

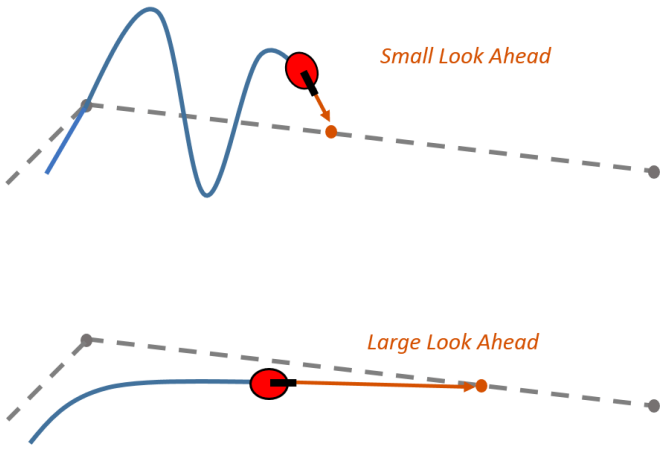


Figure 12: The consequences of incorrect lookahead distances

The method ordinarily employed to combat these inaccuracies is to make the lookahead distance a function of the entity's velocity [3]. This way, a fast moving entity will not fall prey to oscillating around the path, and slow-moving vehicles will have enough information to accurately follow their route. The addition of velocity works very well for modern versions of algorithms such as Pure Pursuit, which work by 'geometrically calculating the curvature of a circular arc that connects the rear axle location to a goal point on the path ahead of the vehicle' [19]. Unfortunately, we are unable to work with trajectories, and the velocities we have are calculated as a function of a the waypoint distance, so this approach will not be possible. However, in the next chapter I will explain an alternative way of combating the issue.

4.5.1 Variation of lookahead distance

My initial thought was to simply use the lookahead distance given in the CARLA simulators pre-set splines ($\sim$2 metres), with logic being that the controllers would perform best, given that was the lookahead distance in mind when they were designed. However, we must remember that this converter is intended to produce good results for several simulators. Since we are unable to make the lookahead distance a function of velocity, it is difficult to find a reliable way to vary the lookahead distance within the path. However, it is possible to find the best lookahead distance on average for a given scenario. This can be done by producing several output files, each with differing lookahead distances (ranging from 1-5m), and then comparing these to the original scenario to see which is the most accurate. It is now not enough to analyse the results by eye, we need a concrete, analytical measure of the converted scenarios accuracy compared to the logs it's based on.

4.6 Scenario Comparison

Since all we have of the original scenario are the logs, the first challenge is to create logs of a similar type for the re-ran simulation. This way, we can just compare various aspects of the log file as a measure of difference between the two. There already exists such a tool for this purpose, developed by A. Ghorbrial and found here [18]. Running this script alongside a scenario produced by my converter records all the necessary information at different time intervals, in a manner extremely similar to the produced logs. To analyse the accuracy of the converter, I will compare the positions in the log files at matching times and record the difference.

Unfortunately, there is one slight difference in the produced logs. The game-produced logs were created as part of an *asynchronous* execution, whereas CARLA runs *synchronously*. In essence, this means that when the initial scenario was produced, it was done so in an environment where a process could be executed before the previous command fully executed [15]. As such, when the call to log the scenario details every x seconds is made, the data is logged exactly then. Conversely, when logging the re-created scenario, there is a delay between the call to log and the actual data being read, due to CARLA finishing the processing of other areas before taking the data from the simulation.

As a result, we end up with a log file that looks something like:

```
testNo ,repeatNo, agentNo, agentID, agentType, agentTypeNo, x, y, z, yaw, vel_x, vel_y, vel_z, speed, time, sim_time, fps
0, 0, 529, 529, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.743, -2.094, 0.000, 0.000, -0.148, 0.148, 0.000, 37.855, 0.000
0, 0, 530, 530, walker.pedestrian.0002, 1, -22.240, -62.490, 1.893, 0.000, 0.000, 0.000, -0.148, 0.148, 0.000, 37.855, 0.000
0, 0, 529, 529, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.663, -2.094, 0.000, 0.000, -1.222, 1.222, 0.110, 37.965, 0.000
0, 0, 530, 530, walker.pedestrian.0002, 1, -22.240, -62.490, 1.818, 0.000, 0.000, 0.000, -1.222, 1.222, 0.110, 37.965, 0.000
0, 0, 529, 529, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.481, -2.094, 0.000, 0.000, -2.230, 2.230, 0.213, 38.068, 0.000
0, 0, 530, 530, walker.pedestrian.0002, 1, -22.240, -62.490, 1.640, 0.000, 0.000, 0.000, -2.233, 2.233, 0.213, 38.068, 0.000
0, 0, 529, 529, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.186, -2.094, 0.000, 0.000, -3.265, 3.265, 0.319, 38.174, 0.000
0, 0, 530, 530, walker.pedestrian.0002, 1, -22.240, -62.490, 1.348, 0.000, 0.000, 0.000, -3.270, 3.270, 0.319, 38.174, 0.000
0, 0, 529, 529, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 0.735, -2.094, 0.000, 0.000, -4.401, 4.401, 0.435, 38.290, 0.000
0, 0, 530, 530, walker.pedestrian.0002, 1, -22.240, -62.490, 0.958, 0.000, 0.000, 0.000, 0.000, 0.000, 0.435, 38.290, 0.000
```

Figure 13: Logs produced by gathering data from CARLA simulation

These are identical aside from the time steps of the data being recorded. As you can see, instead of regular steps, the data is stored at times such as 0.110, 0.213 etc. Note that these values are arbitrary and will differ with each run of the OpenScenario file, another consequence of the non-deterministic nature of CARLA. Clearly, we cannot compare the positions at these differing times, and as such must employ some interpolation strategy.

4.6.1 Interpolation

Since I would prefer the analysis tool to stand separate from the converter, and only analyse between the two logs, it cannot know the location of waypoints in the re-created simulation, as these are stored in the OpenScenario file. As such, there is no way infer what speed the vehicle was travelling at aside from once again resorting to distance/time. Given the irregular time spacing between records in the re-created logs, it is more consistent to interpolate from our initial

logs. In other words, if I am given a simTime value of 0.213, rather than calculate where this entity should be at 0.200, it is less error-prone if I instead interpolate the position of the original scenario's entity at 0.213 from 0.200. I argue this is a more accurate approach as the average speed will always be calculated over the distance travelled in 0.1 seconds, and so cannot lead to a large inaccuracy, whereas the inconsistent times found in the re-created logs could have gaps of up to $\sim$0.2 seconds, where averaging could have more impact.

To further clarify this method of interpolation, I will give an example. Say we have three log entries, two from the game-produced logs, the other gleaned from CARLA running a generated OpenScenario file (for the purpose of this example, only distance and time will be given).

- Game Logs1 - x = 4.800, y = -2.931, z = 153.837, simTim = 0.700
- Game Logs2 - x = 4.988, y = -2.931, z = 154.983, simTim = 0.800
- CARLA Logs - x = 4.942, y = -2.932, z = 155.253, simTim = 0.774

My code will identify that 0.774 is between 0.7 and 0.8, then calculate the average speed per co-ordinate direction the entity was travelling at between these two points in time. So here for x:

$$xVelocity = (4.988 - 4.8)/0.1 = 1.88m/s \tag{1}$$

We then find where the game logs would be at 0.774 seconds in time.

$$xPosition = 4.8 + (1.88 * 0.074) = 4.939 \tag{2}$$

Doing the same for x and y grants values yPosition = -2.931 and zPosition = 154.685.

We can now compare these values to those found in the CARLA logs. Doing this, and finding the difference between values for each co-ordinate and then adding these gives a sum of the inaccuracy found at this point in time. In reference to our example, we find at 0.7 seconds into the simulations, the position in the re-created CARLA scenario is incorrect by 0.572m. Running this method for every line of the CARLA logs will give us a measure of how accurately the scenario has been recreated, which could be divided by the number of logged values for an average inaccuracy per log.

There are some things to note about this measure of inaccuracy that you should keep in mind when comparing the values produced between scenarios. It should not be used as a reliable, precise source of error, since there are approximations in the interpolation, and seeming inaccuracies even at the 0.00 of the log files, with entities stated to spawn in slightly the wrong location. As a result, the difference between the different logs will likely be quite high, as there are multiple sources of uncertainty (and the deviations we expect due to CARLA). Due to this, it is difficult to draw conclusions on how large and where the errors in the OpenScenario files are. However, it does provide us with a useful

metric for comparing several different versions of a re-generated scenario. With this in mind, it is now possible to vary several parameters (most obviously the lookahead distance) and compare the performance of the converter with these different values.

Here are the key code snippets from the variance measurement tool, with the coVel class simply storing the 3 co-ordinate velocities at a particular point in time:

```
private void calculateVariance() {
    for (int i = 0; i < simLogs.size(); i++) {
        int lineNo = findMatchingLine(simLogs.get(i));
        if (lineNo == -1) {
            return;
        }
        coVel avgVel = calculateAverageSpeed(gameLogs.get(lineNo),
            gameLogs.get(lineNo+entityNo));
        double timeDiff = simLogs.get(i).getSimTime() -
            gameLogs.get(lineNo).getSimTime();
        double newX = gameLogs.get(lineNo).getX() +
            avgVel.getX()*timeDiff;
        double newY = gameLogs.get(lineNo).getY() +
            avgVel.getY()*timeDiff;
        double newZ = gameLogs.get(lineNo).getZ() +
            avgVel.getZ()*timeDiff;
        variance = variance + (Math.abs(newX-simLogs.get(i).getX()));
        variance = variance + (Math.abs(newY-simLogs.get(i).getY()));
        variance = variance + (Math.abs(newZ-simLogs.get(i).getZ()));
        count++;
    }
}
```

```
private int findMatchingLine(logLine line) {
    for (int i = 0; i < gameLogs.size() - entityNo; i++) {
        if (gameLogs.get(i).getSimTime() <= line.getSimTime() &&
            line.getSimTime() <=
            gameLogs.get(i+entityNo).getSimTime()) {
            if
                (gameLogs.get(i).getAgentType().equals(line.getAgentType()))
                {
                return i;
            }
        }
    }
    return -1;
}
```

```
private coVel calculateAverageSpeed(logLine source, logLine
```

```
     destination) {
    double xDiff = Math.abs(source.getX() - destination.getX());
    double yDiff = Math.abs(source.getY() - destination.getY());
    double zDiff = Math.abs(source.getZ() - destination.getZ());
    double timeDiff = destination.getSimTime() - source.getSimTime();
    coVel avgSpeed = new coVel(xDiff/timeDiff, yDiff/timeDiff,
        zDiff/timeDiff);
    return avgSpeed;
}
```

5 Analysis

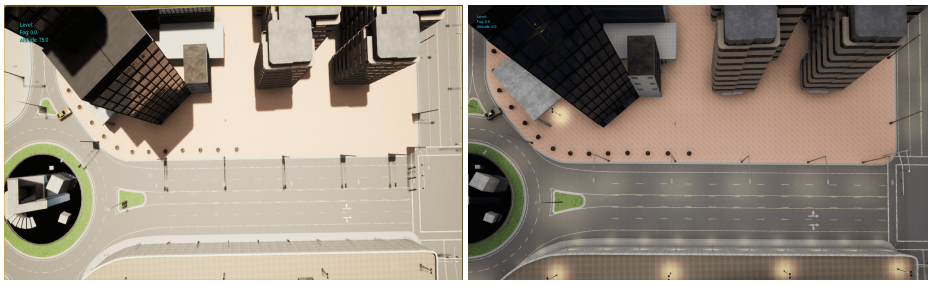
5.1 Introduction

With the development of the converter and relevant analytical tools completed, we are now able to scrutinise the resulting logs and scenarios. This section will cover the accuracy of the converter, and where it falls short, suggest reasons for this and potential solutions. I will also investigate the effects that varying certain parameters in the OpenScenario file has on the scenarios variance and once more try to justify this.

5.2 Converter Performance

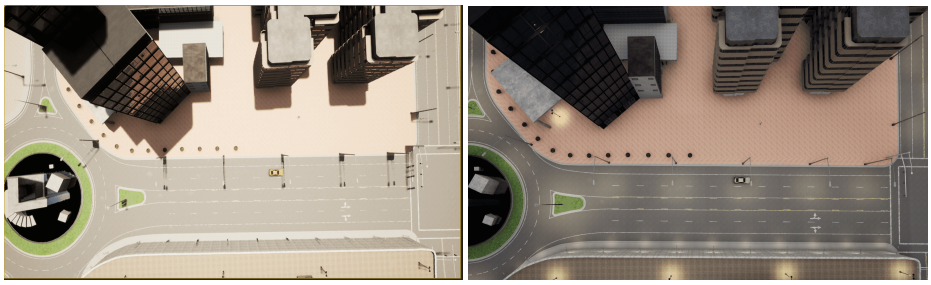
Perhaps the most intuitive way to measure the performance of a scenario converter is to inspect its produced scenarios with the human eye. While obviously this is not capable of spotting slight errors or changes in the scenario, it is a good starting point. For select examples, rather than working with logs from the game, I instead chose to work with manufactured CARLA scenarios, with the same log format, but the addition of a video to show how the scenario would look. Since CARLA is the primary simulator I chose to work with, the vast majority of my recreated scenarios were also run on this simulator. As such, if the converter did indeed work, the video I have associated with the initial logs should match what runs when I executed a converted scenario almost exactly (bearing in mind certain limitations I will later explore). This method of analysis was mainly used throughout the construction of the converter as a means for me to test whether or not my code was working as intended, but does also serve to prove empirically that the converter was at least somewhat successful.

Since most PDF viewers do not support embedded video playback, I shall give an example of this with reference pictures of both the initial and converted scenarios at key points, and the accuracy can be inferred from the relative positions of the actors. The original (day) is on the left and the converted (night) is on the right.



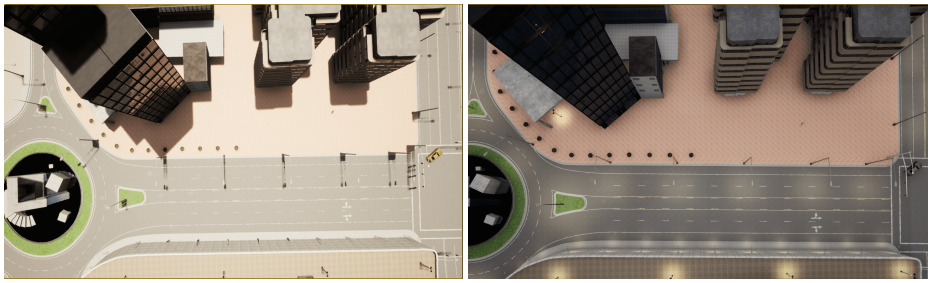
(a) Initial scenario - start (b) Converted scenario - start

Figure 14: A comparison of the opening scene of a scenario and its converted partner



(a) Initial scenario - ongoing (b) Converted scenario - ongoing

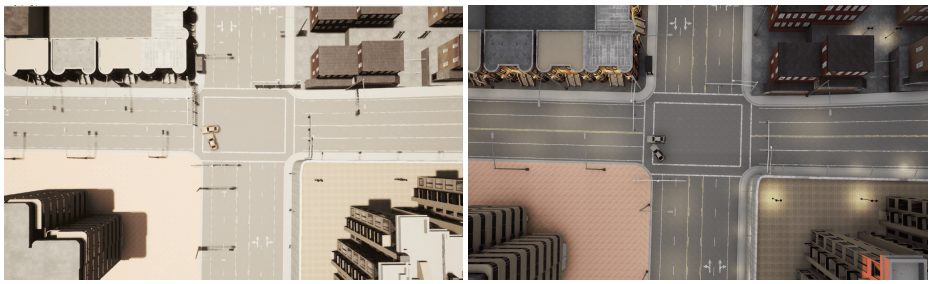
Figure 15: A comparison of the middle of a scenario and its converted partner



(a) Initial scenario - near scenario end (b) Converted scenario - near scenario end

Figure 16: A comparison of the ending moments of a scenario and its converted partner

These images show that the converter does indeed re-create scenarios without a collision rather well. It also is able to re-create those with collisions, although in a much less reliable manner. Here another image comparison showing that the point of contact during a collision remains almost completely correct when converted (With lookahead distance in mind):



(a) Initial scenario - collision (b) Converted scenario - collision

Figure 17: A comparison of the opening scene of a scenario and its converted partner

Note that the trajectories in both these scenarios are recreated similarly in esmini, another simulator designed solely to run OpenScenario files. This satisfies our goal of ensuring that the converter works across various platforms. The only difference required is the removal of the CARLA: in the file heading, since this alters how co-ordinates are interpreted. However, it is very difficult to show this success since only the OpenDRIVE files are used, without any other graphical map information, leading to the pedestrian seemingly floating in the void, and no easy reference points to judge the vehicles movement by.

The issue we see with crashes (much more apparent in the video), is that the vehicles seem to slow down before they make contact. This is no illusion, and is due to the way that the converter is coded. Since the speed of the vehicle between two waypoints is dictated by the average speed between these distances in the initial log file, if a crash occurs in this space, the average speed value will not properly reflect the collision. For example, say we have two vehicles driving towards each other at 10m/s, and then upon colliding drop to 0m/s, before slowly moving off again at say 2m/s. In the converted scenario, instead of the vehicles aiming to collide at 10m/s, it will actually be noticeably less than this because of the influence of the time stopped coming into play in the average speed calculation. This also slightly changes the position of the vehicle impact due to the cars slowing down sooner. In some cases, the crash is missed completely.

This is very difficult to tackle given the tools we have to work with. Ideally, we could set the speed at frequent intervals, but having tried this it does not work. This is due to two reasons, firstly, having so many sample points leads to erratic movement of the vehicle due to constantly changing speeds in an abrupt manner. More importantly than this, the size of the vehicle's BoundingBox (detection area) would cause CARLA to try and apply several speeds at once, at worst causing errors, at best taking the speed from the last waypoint within detection. Since it doesn't seem possible to fix this issue, we will instead attempt to minimize it by reducing the impact of average speeds.

5.2.1 Parameter Experimentation

As previously mentioned in section 4.5.1, for the scenario with a crash, I have produced several OpenScenario files, each with varying lookahead distances. What we're looking to optimize here is the trade-off between turn accuracy and collision speed. The logic behind this is that, the larger the lookahead distance, the greater the distance between waypoints. As such, over a larger distance, the impact of a brief period where entities stop moving will have less of an effect. This means that the vehicles will collide at a more accurate velocity. However, if the lookahead distance is too great, we risk not expressing the true path of the vehicle. This will be most noticeable during turns, where if the distance is too high, many turns will be completely skipped in favour of a more direct route.

More formally, here we will vary the path generated by the converter through alteration of the waypoint calculation logic, and assess which of these paths is optimal in terms of recreating the logged scenario. The metric used to determine optimality will be average deviation per recorded time-step between the game logs and the re-created scenario. While this measure may not produce the best lookahead distance in terms of collision recreation, it is important that the scenario accuracy for the rest of the simulation is not disregarded.

With this in mind, I hypothesise that the optimal lookahead distance for the majority of scenarios will be relatively low in order to preserve small movements of the vehicle, but may not be as low as possible where collisions are involved due to the reasons discussed above.

To test this hypothesis, for the crash scenario pictured in the previous section, here is a graph displaying the variance of the logged simulation (Average total distance error per scenario log per entity) against the lookahead distance used in its creation.

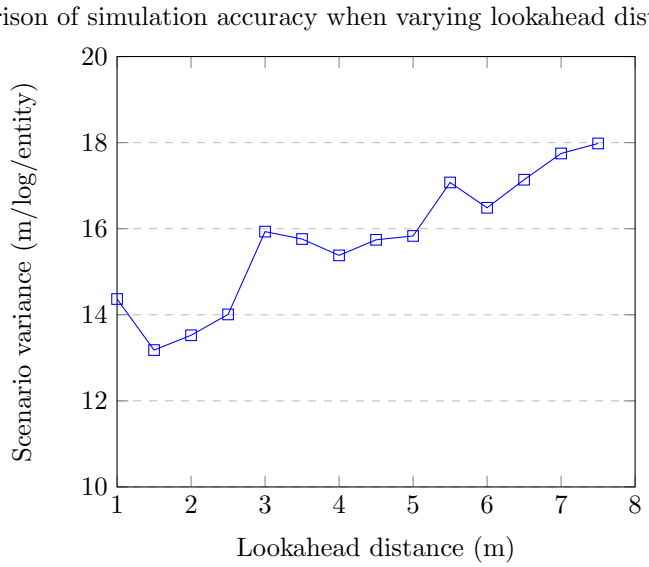


Figure 18: A comparison of simulation accuracy against lookahead distance (collision)

This graph showcases interesting results. It claims that the accuracy of the simulator continues to increase as you lower the lookahead distance, provided that you do not go beyond the entity length as mentioned above. This shows that the speed at which collision occurs is not the most important factor in re-creating a scenario accurately. It also suggests that other factors such as the angle and position of the collision may be more important. Since this tool is incapable of judging accuracy during small portions of the simulation, it is also plausible that the crashes are less accurate but this is made up for by precise path movement for the rest of the scenario recreation.

As well as this, it is important to note that this optimal lookahead distance may not be consistent across all scenarios. To find the best parameter settings for a particular scenario, I would recommend using this tool to narrow down the range of values, and then undergo visual analysis to ensure the key scenes are carried out correctly.

We can also speculate as to why this inaccuracy seems so high even in the best case scenario. The most likely explanation for this is that the addition of the logging process causes extra strain on my computer, and causes system resources to be allocated away from Unreal Engine. As mentioned in section 4.3.2 (Non-determinism), this is one of the two direct causes of variance when running the same scenario. Additionally, The initial z-positions of vehicles do not seem to align, which could indicate that either the entity has moved before

the initial log is taken, or that there are some inaccuracies in the logging tool. At first, I assumed this was due to one of the vehicles spawning on a hill, but this inaccuracy is maintained in other scenarios without uneven terrain. This leads me to believe that the additional strain of logging alongside simulating causes some kind of drift at scenario start, while the scenario is initialising.

For reference, here are the differing log files at time 0.00 (A scenario on even terrain):

```
testNo, repeatNo, agentNo, agentID, agentType, agentTypeNo, x, y, z, yaw, vel_x, vel_y, vel_z, speed, time, sim_time, fps
0, 0, 1, 1679, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.545, -2.094, 0.000, 0.000, -2.935, 2.935, 0.000, 978.104, 0.000
0, 0, 2, 1680, walker.pedestrian.0002, 1, -22.240, -62.490, 1.694, 0.000, 0.000, 0.000, -2.450, 2.450, 0.000, 978.104, 0.000
```

(a) Initial logs

```
testNo ,repeatNo, agentNo, agentID, agentType, agentTypeNo, x, y, z, yaw, vel_x, vel_y, vel_z, speed, time, sim_time, fps
0, 0, 89, 89, vehicle.mercedes-benz.coupe, 3, -26.260, -7.950, 1.742, -2.094, 0.000, 0.000, -0.211, 0.211, 0.000, 124.686, 0.000
0, 0, 90, 90, walker.pedestrian.0002, 1, -22.240, -62.490, 1.892, 0.000, 0.000, 0.000, -0.211, 0.211, 0.000, 124.686, 0.000
```

(b) The recreated log

Figure 19: Discrepancy between the log files at 0.00

To narrow down how much how much of the error in recreation is down to the collision, I produced a similar graph for a simpler scenario with no collision involved.

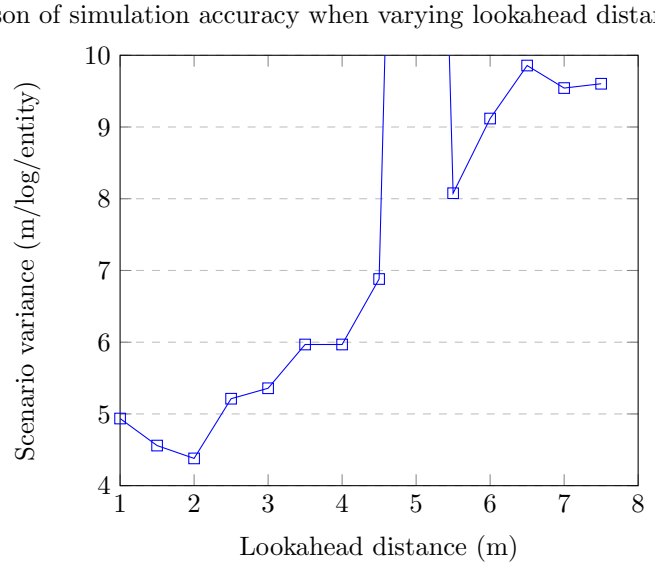


Figure 20: A comparison of simulation accuracy against lookahead distance (No collision)

We see a similar trend emerge, with lookahead distances in the 1-2m range

performing far better than when increased. The anomaly at 5m was due to an entity getting stuck against a static object for a large portion of the scenario. This graph further supports the theory that for most cases the optimal lookahead distance seems to lie within this 1-2m range. More importantly, it verifies that a large portion of the scenario inaccuracy is caused by vehicles colliding, with the converter accuracy being 3 times as high in comparison.

5.3 Summary

To summarise, bare-bones visual analysis shows that the converter is able to re-create inputted driving scenarios to be easily recognisable to the human eye. This remains true for scenarios where collisions are present. However, upon closer inspection through analytical tools, the accuracy of the converter leaves much to be desired, especially during and after a crash. In attempt to improve this, the lookahead distance was varied with the goal of finding the optimal entity paths. I found that this tends to lie around the 2m mark, with sharp decreases in performance at higher vales.

6 Evaluation

6.1 Suitability

With the converter complete, we can now assess the suitability of OpenScenario in creating the converter, and the converter itself in achieving its purpose.

6.1.1 OpenScenario

As I delved further into OpenScenario, I realised that it excels in describing scenarios where maneuvers are enacted based on the position (oftentimes relative) of other entities. For instance, an attempted overtake being aborted due to an oncoming car, or a pedestrian only choosing to cross the road when it is deemed safe. Unfortunately, to fully utilise these options, prior knowledge of the scenario is necessary. It is impossible to express possibilities in a format as primitive as log files, either the overtake is successfully completed based on the co-ordinates, or a collision occurs. Given that the generality of the converter is essential, many of the more advanced features of OpenScenario cannot be used.

An example of an advanced feature unavailable for use is the SyncrhonizeAction. This allows vehicles to arrive at the same point simultaneously, with a set speed if required. Collisions coded in this way would be extremely consistent, as the simulator itself is responsible for their meeting. Once more this would also require prior knowledge that this scenario results in a collision, and where this crash takes place.

The other issue, already somewhat discussed, is the limitations in simulators that implement OpenScenario. This further restricts the Actions you can use, and impacted the converter heavily, mainly in terms of accurate speed calculation (Though it could be argued that this is a fault of the log file for not supplying this information). This issue will overtime become less prevalent as the simulator developers add more and more functionality.

Nonetheless, despite these issues, OpenScenario is certainly suitable. The fact that it is a universal standard is essential for this project, and as of this moment, no alternative exists. As for the missing advanced features, this is not OpenScenario being the wrong tool for the project, more that it is capable of everything within this papers scope and more. Potential future work involving more features will be discussed in a later section.

6.1.2 Converter

At the onset of this project, I looked to research a way to increase the verification coverage of AVs in a simulated environment by taking the output of a simple game and confirming that it could be accurately re-created in a more complex environment. To assess whether the converter is deemed a promising avenue in this regard, we need to look back at the concept of tolerance (Non-determinism, Section 4.3.2). Referring back to the paper that introduced the concept of

tolerance, the general limit of variance in repeated scenarios is 1cm [11]. In other words, if repeated runs of the same scenario differ by more than 1cm at any point, we cannot consider them to have increased verification coverage. This ensures the high standards of AV safety are met.

Taking this into account, it is clear that the converter cannot increase coverage in a directed way. By this I mean that the variance between the logged data and produced scenario is far greater than 0.1cm. This is seen in the two graphs above, with the lowest average deviation from the logs being roughly 4m, implying that at the very least 1 of the points an entity is greater than 0.1cm away from its position (with an extreme probability that most points are). However, this does not mean that verification coverage cannot be improved at all. Despite the fact that the re-created scenarios are not precise enough to be considered direct copies of the inputted data for the purpose of verification, they may stand as their own scenarios. If this scenario is similar enough that it remains useful, and the deviation between several runs of the new scenario is less than 0.1cm at all points, then verification is still possible.

To test this, I will run the analysis tool, but instead of the arguments being a game log file and simulated log file, we compare several runs of the same simulated OpenScenario file. Running the scenario analysed in the figure 18 5 times, we then compare the logs each of these produced with each other, with the results shown in a table below: (Note that analysing 1 vs 2 and 2 vs 1 produce different results due to the interpolation method employed.)

Simulation Iteration	Test 1	Test 2	Test 3	Test 4	Test 5
Test 1	N/A	1.953	2.311	2.506	2.812
Test 2	2.625	N/A	0.377	0.617	0.730
Test 3	2.979	0.754	N/A	0.405	0.481
Test 4	3.184	0.920	0.543	N/A	0.267
Test 5	3.470	1.235	0.871	0.668	N/A

Figure 21: A table showing the average variance of repeated simulation runs

Similarly to above, all scenario comparisons result in an average deviation of greater than 0.1cm, though some figures are close! This means that the tool as it stands is unable to provide the repeat-ability necessary to confidently increase AV coverage. Although it is the shortcomings of the converter that make directed verification impossible, this failure lies in the tools used. Simulating a scenario based on the same OpenScenario file can lead to results as anomalous as Test 1, with average deviance as high as 3.470. This speaks volumes about the impacts of non-determinism, and the inaccuracies caused by the interpolation of the log files. As such, I still feel the converter is a feasible way to produce scenarios with tolerable deviance, however CARLA (run on a home PC) is not the environment to achieve this. Alternative paid tools, such as RFpro, boast a

deterministic environment which would elimination variation and also allow for an easier logging process with use of the 'fixed timestep simulation mode' [26].

6.2 Objective Achievement

To further evaluate my project, let us discuss whether or not all the goals set were met.

For reference, here are the aims, which I will evaluate one by one.

1. Research the OpenScenario file type to determine whether it is possible to convert the data in the logs to a complete OpenScenario file, capable of re-creating the logged simulation. If yes, determine the limitations of the format.

2. Create a conversion tool that, when given this raw data about agents in a scenario, is able to convert this into an OpenScenario file that can be re-created across various simulators. Explore the accuracy of this converter, especially when a collision is involved.

3. After analysis, attempt to improve the performance of the converter, and then draw conclusions about the efficacy of this conversion method.

6.2.1 Aim 1

As shown by the work and analysis above, producing the converter was indeed possible, and it is capable of re-creating physically sensible scenarios, maintain the key scenes. The main thing to evaluate here is discussion of the 'limitations of the format'. These are fully explored above, and also discussed briefly in section 4.3.

6.2.2 Aim 2

Once again, the converter stands as proof, with the accuracy explored both empirically and through use of the developed deviation measurement tool. This tool is used to specifically quantify the impact collisions have on converter accuracy.

6.2.3 Aim 3

I attempted to optimise the performance of the converter through variation of the lookahead distance, though I could not find any way of modifying the converter code itself to produce results that more accurately mimic the original scenario. This is because the converter design is heavily restricted by CARLA and the log file format. The converter itself was analysed in-depth above.

6.3 Alternative Direction

While undertaking this project, I developed a more complete understanding of the areas surrounding AVs, and a greater appreciation for the problems in the industry that need to be tackled. Not only this, the process of researching simulation/verification and developing/analysing the converter gave rise to several related ideas that could make interesting follow-up projects.

6.3.1 Collisions

When attempting to re-create collisions from the game logs into a more complex simulator, I found myself thinking it somewhat backwards to try and perfectly mimic the result of the game's collision logic in a more advanced simulator. Surely, if the goal is to increase verification coverage, then we should entrust the collision logic to the most accurate software in terms of physics. In addition to this, the issue of collision speed with averages was a large obstacle. With this in mind, I propose an idea that leaves the collision in the hands of your simulator of choice with the correct speed.

Once again, armed with only log files, we could develop code to determine whether a collision has taken place in the scenario, given knowledge of the vehicle model sizes and their co-ordinates. If a collision did occur, we could then cut the log file just before that point. This way, when creating the resulting OpenScenario file, the final waypoint would be just before the vehicles should collide. As such, the average speed calculation would not be affected by the crash at all. Once the vehicles reach the final waypoint, they could then be instructed to continue on their current trajectory with the same speed, leading to a much more accurate collision.

The effectiveness of this idea would vary from simulator to simulator, but could serve to increase the verification coverage of scenarios where a collision is present. I would once again expect the best results through the use of a deterministic simulator.

6.3.2 Other Uses

It is also worth bringing into question the fundamental principle that converters much like this should maintain the ability to take in any kind of scenario. While this converter attempted to bring added value through the ease of access in terms of scenario generation and its ability to convert anything, these need not be the cornerstones of other projects. For example, should a niche converter be designed which can only convert overtaking scenarios, although less general, this could still greatly speed up the tedious process of OpenScenario file writing while using more of the formats functionality. While this would require the person using the converter to be aware of the its limitations, scenarios could still be created without expert knowledge. In other words, now that conversion to OpenScenario through an automated tool is shown to be possible, it

would be interesting to experiment with what could be implemented into these OpenScenario files if restrictions were placed on the scenario.

6.4 Future Work

In terms of work aimed at expanding this particular project, time will be a major factor. The more support that is given to OpenScenario, the more accurate projects in this vein can be. As soon as FollowTrajectoryAction is implemented in CARLA, many issues facing collision recreation will be greatly diminished. More simulators will continue to adapt the OpenScenario schema into their products, and open-source competitors to CARLA will emerge. Outside of waiting for these features, a major boon to this work would be the improvement of the logger and interpolation methods. With more advanced logging and comparison, it would be far easier to determine the inaccuracy generated by the conversion itself. With this knowledge, it would be possible that some scenarios produced by the converter would be stable enough to act as a verification method.

The method of log generation could also be experimented with, since more information can never be a bad thing. Example features could be accurately recording the velocity, or properly recording the placement of static actors as well dynamic entities. With the addition of these, modification of the OpenDRIVE file type would enable an even wider range of potential scenarios. The modification of OpenDRIVE files is no easy feat however, and would most likely constitute a project in and of itself.

7 Conclusion

Through the research carried out in this project, we can conclude that the potential of a first of its kind tool which converts raw scenario data to a file type such as OpenScenario is huge. The ability to transform simple data to a human and machine-readable scenario description is extremely valuable, even in areas outside this project's scope. The development of this new tool was achieved here, and is capable of recreating inputted scenarios to a degree of accuracy which satisfies the human eye.

However, once analysed numerically, it became clear the the converter as it stands cannot fulfill its intended purpose. The level of uncertainty involved in the converter itself and associated tools give rise to a variance unacceptably high if we aim to increase verification coverage for autonomous vehicles. Regardless, this project serves as a proof of concept that the conversion between game logs and OpenScenario files can indeed by carried out, and paves for way for future applications using this tool, with some suggested above.

The causes of the developed tool's inaccuracies have also been investigated throughout, and found many of these could be significantly reduced given further advancements in the technology used and their level of support for one another. From this, I am confident that, given time and access to more advanced software, a very similar tool could be developed. This next iteration would have levels of accuracy high enough to be used in helping get autonomous vehicles on the road more quickly and safely.

References

[1] R. Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm.* Tech. rep. Jan. 1992.

[2] A. Engel. *Verification, Validation, and Testing of Engineered Systems.* John Wiley & Sons, Incorporated, 2010, preface.

[3] H. Chang-Soo K. Dong-hyung and L. ji. "Sensor-based motion planning for path tracking and obstacle avoidance of robotic vehicles with non-holonomic constraints". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 227 (Jan. 2013), pp. 178–191. DOI: 10.1177/0954406212446900.

[4] M. Palmieri. "System Testing in a Simulated Environment". MA thesis. Mälardalen University, Apr. 2013.

[5] Simon Ulbrich et al. "Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems.* IEEE, 2015.

[6] M. Wagner P. Koopman. *Challenges in Autonomous Vehicle Testing and Validation.* Tech. rep. Jan. 2016.

[7] Jan-Pieter Paardekooper Erwin de Gelder. "Assessment of Automated Driving Systems Using Real-Life Scenarios". In: *2017 IEEE Intelligent Vehicles Symposium.* Redondo Beach, CA, USA: IEEE, 2017.

[8] C. Medrano-Berumen et al. C. Stark. "Generation of Autonomous Vehicle Validation Scenarios Using Crash Data". In: *SoutheastCon 2020.* Raleigh, NC, USA: IEEE, 2020.

[9] M. Wen J. Park & K. Cho. "A scenario generation pipeline for autonomous vehicle simulators". In: *Human-centric Computing and Information Sciences* 10.24 (2020).

[10] A. Ghorbrial et al. K. Eder. "An Agency-Directed Approach to Test Generation for Simulation-based Autonomous Vehicle Verification". In: *Annalen der Physik* (2020), pp. 31–38.

[11] A. Ghorbrial et al. G. Chance. "On Determinism of Game Engines used for Simulation-based Autonomous Vehicle Verification". In: (2021).

[12] ASAM. *ASAM OpenSCENARIO: User Guide.* URL: https://releases.asam.net/OpenSCENARIO/1.0.0/ASAM_OpenSCENARIO_BS-1-2_User-Guide_V1-0-0.html. accessed: 30/08/2021.

[13] BBC. *The Highway Code.* URL: https://www.gov.uk/browse/driving/highway-code-road-safety. accessed: 29/08/2021.

[14] CARLA. *OpenSCENARIO support - CARLA ScenarioRunner.* URL: https://carla-scenariorunner.readthedocs.io/en/latest/openscenario_support/. accessed: 31/08/2021.

[15] R. Costa. *Asynchronous vs. Synchronous Programming: When to Use What.* URL: https://www.outsystems.com/blog/posts/asynchronous-vs-synchronous-programming. accessed: 04/09/2021.

[16] C. Criddle. *'Self-driving' cars to be allowed on UK roads this year.* URL: https://www.bbc.co.uk/news/technology-56906145. accessed: 22/08/2021.

[17] *Definition of scenario.* URL: https://www.lexico.com/definition/scenario. accessed: 25/08/2021.

[18] A. Ghorbrial. *github - logging function.* URL: https://github.com/Abanoub-G/carla_0.9.11/blob/master/PythonAPI/examples/TestBench/just_logging.py. accessed: 04/09/2021.

[19] S. Kundu. *Understanding Geometric Path Tracking Algorithms — Stanley Controller.* URL: https://medium.com/roboquest/understanding-geometric-path-tracking-algorithms-stanley-controller-25da17bcc219. accessed: 01/09/2021.

[20] MathWorks. *Automated Driving Toolbox.* URL: https://uk.mathworks.com/products/automated-driving.html#resources. accessed: 29/08/2021.

[21] MathWorks. *Export Driving Scenario to OpenScenario.* URL: https://uk.mathworks.com/help/driving/ref/drivingscenario.export.html. accessed: 29/08/2021.

[22] MathWorks. *Lidar processing.* URL: https://uk.mathworks.com/help/driving/lidar-processing.html. accessed: 29/08/2021.

[23] Mathworks. *Pure Pursuit Controller.* URL: https://uk.mathworks.com/help/robotics/ug/pure-pursuit-controller.html#burwbfa. accessed: 01/09/2021.

[24] Merriam-Webster. *Definition of crossfall.* URL: https://www.merriam-webster.com/dictionary/crossfall. accessed: 29/08/2021.

[25] MITRE. *VERIFICATION AND VALIDATION.* URL: https://www.mitre.org/publications/systems-engineering-guide/se-lifecycle-building-blocks/test-and-evaluation/verification-and-validation. (accessed: 13/08/2021).

[26] R. Smith. *Simulation Control for Vehicle Automation.* URL: https://www.claytex.com/tech-blog/simulation-control-for-vehicle-automation/. accessed: 08/09/2021.

[27] *What is System Testing? Types & Definition with Example.* URL: https://www.guru99.com/system-testing.html. (accessed: 13/08/2021).

[28] Wikipedia. *OpenDRIVE(specification).* URL: https://en.wikipedia.org/wiki/OpenDRIVE_(specification). accessed: 30/08/2021.