Passau University

Master Thesis

DriveCloud
A cloud based Infrastructure for Testing Autonomous Vehicles

Author:
Stefan Huber

Supervisor:
Prof. Dr.-Ing. Gordon Fraser

Advisor:
Alessio Gambi, Ph.D.

Tuesday 26th September, 2017

Abstract

Testing autonomous cars is even when based on simulations very time consuming. This results from the time the simulation itself needs, the time needed for setting up the simulation environment, defining test cases, evaluating them and often a lack of computational power. To address this I propose a cloud based system for testing autonomous cars tackling these problems. Such an infrastructure will significantly reduce the time needed to spend on test setup, execution and evaluation. Furthermore it will allow to share computational resources among many other researchers.

1 Introduction and Motivation

Is to much for an introduction? It seems like it already contains stuff belonging to background, state of the art or proposed method section. The great progress made over the past years in developing autonomous vehicles and extending their capabilities goes hand in hand with a drastically increasing need of testing them. Since the number of possible test cases is infeasible to test there are various approaches which try to reduce the number of scenarios necessary to test by searching for very interesting ones including test cases which would be very dangerous for any participant in daily life or which represent edge cases. E. g. [3] quantifies how interesting test cases are by measuring their criticality. In terms of the paper the criticality is described by the area a car is able to maneuver in without crashing. The approach in [17] creates interesting test cases by generating tracks. Therefore genetic algorithms are used. Other approaches like [6] reconstruct test cases based on real accidents assuming these are interesting.

Nevertheless to have a high confidentiality about the safety of autonomous vehicles they still would have to drive millions of kilometers on the highway and in the cities without having an accident. Additionally each time something of the vehicles system changes the process would have to start from the beginning. Furthermore testing cars within daily life would endanger pedestrians and any other traffic participant. As a consequence simulations are widely used which is in combination with the high number of test scenarios still very time consuming and costly. An extensive use of simulation requires a high effort of setting up and connecting software components, lots of hardware infrastructure enabling to handle many simulations simultaneously and a way to express test cases.
The currently most popular formats for describing simulated environments are OpenDRIVE [10] and CommonRoad [2]. Especially OpenDRIVE is very comprehensive concerning its capabilities in describing environments. But both can not contain any test cases. Another approach is Paracosm [18] which is able to phrase environments as well as test cases. The test scenarios even allow reactive definitions of conditions. On the other hand the capabilities of environment description and the compatibility with existing test scenarios are low. So existing and well-tried test scenarios can not be reused.

To cope with all these problems this paper presents an approach for providing a cloud based infrastructure for testing autonomous vehicles and for test case definitions.

1.1 Problem Statement

A very important point of testing is the accuracy of the generated scenarios, the efficiency of the simulator and the physics of all involved objects. To achieve this on a reasonable high level is a very hard and time consuming task. In my work I will use BeamNG [9]. BeamNG tackles these problems tremendously well and provides a research version called BeamNG.research allowing to control simulations using a Python interface.
I may further increase the efficiency of the simulator by allowing a less precise simulation of objects and participants currently not nearby the vehicle under test (VUT) or also referred to as ego vehicle.
The infrastructure of the cloud has to be very scalable in order to handle as many concurrent simulations as possible. Therefore it is essential to implement load balancing to get the maximum performance for each running simulation and elasticity to allow dynamic allocation of additional

resources when the number of simulations to run increases. At this point it is likely that BeamNG needs to be extended or modified in order to use it within the cloud in the desired way. To further increase the utilization of the resources of the cloud process migration of running simulations may be needed.
Also the communication between a simulation and possibly multiple artificial intelligences (AIs) controlling some participants needs to be specified in an efficient and bandwidth saving but yet comprehensive enough way since the simulator has to do lots of requests to AIs controlling traffic participants or the simulation. So it is important to monitor and control the simulation and its communication. The monitored events, test cases and usage statistics should also be collected.

The test case definitions are complex as well. First, the definition of environments is required to be small and reusable but needs high flexibility in terms of the course of roads and the initial states of all involved participants. In order to be able to generate simulations out of OpenDRIVE and CommonRoad environment definitions need to be unified into a common description scheme and then translated into a BeamNG compatible representation. Secondly, the definition of test suites has a very high complexity for having an expressive description which is not only able to express basic conditions like damage, position or speed criteria but also allow restrictions on when certain conditions have to be considered and evaluated. This comes into play if one wants to express conditions e. g. a car must not exceed a certain speed limit while driving on a certain lane. Thirdly, it is not sufficient to define success or failure criteria alone. It is not possible to conclude success (failure) if the simulation does not fulfill some failure (success) criteria.

2 Technical Background and State of the Art

2.1 Technical Background

Both simulations and test cases in my work are based on logical time namely **ticks** [1]. The video showing a simulation in BeamNG has a certain rate of frames per second (FPS) specifying how often BeamNG calculates new positions and properties of any object each second. Each calculation results in a frame part of the video and defines a tick in the simulation time.

Synchronous simulation is an execution strategy dealing with problems of concurrently simulated objects and traffic participants and problems with AIs controlling participants. After each tick the simulation pauses, checks all test criteria and requests all given AIs which control traffic participants and the simulation. Then the next tick is executed and the process starts again.

Elasticity [13] in context of clouds characterizes the ability of the infrastructure to dynamically allocate or free computation resources depending on the current load of the system. A cloud implementing this property is able to deal with sudden increases of executed tests.

An **ontology** [12, 11] is "a specification of conceptualization". This means it is a formal representation of knowledge and its connections. An ontology consists of concepts characterizing classes/types of facts and properties defining which way these classes are connected. E. g. a concept "lane" may have properties like speed limit, width, direction, position and another property stating that this lane intersects with another lane. *TODO Improve definition*

2.2 State of the Art

Does this section duplicate stuff from the introduction section?
One of the most popular description schemes of simulation environments is OpenDRIVE [10]. It allows to define extremely comprehensive and very detailed road networks including predecessor/successor roads, junctions, neighbor lanes, corner roads, side walks, cross falls, parking spaces, bridges, signals, signes, railroads, markings and thousands of objects and participant types more. This information can not be fully utilized since the BeamNG or at least its interface does either not allow or is not capable of encode all this information. So in my work I only want to consider smaller subsets of this scheme.

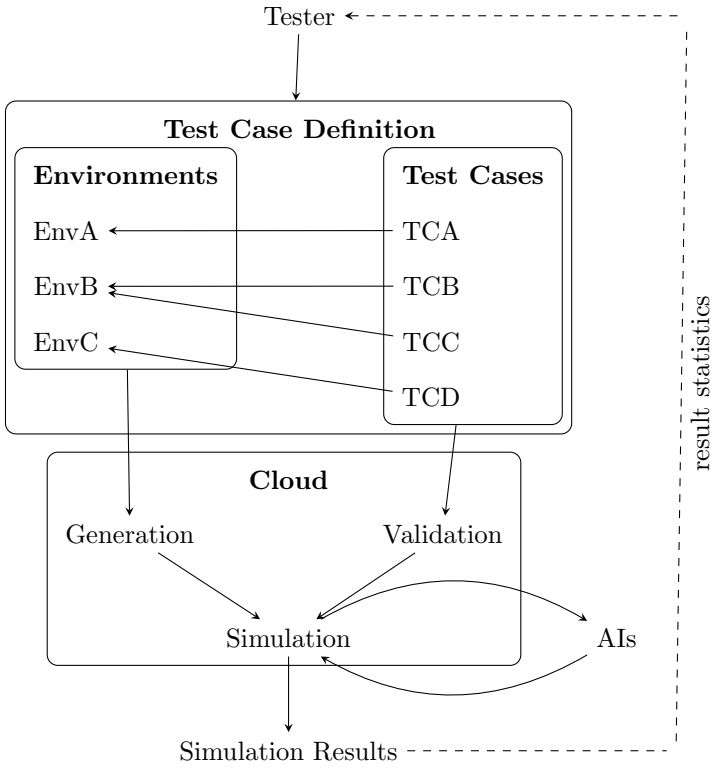
CommonRoad [2] is very restrictive in its capabilities in comparison to OpenDRIVE and focuses solely on the description of simple lanes, basic participants as well as obstacles in the two dimensional space. CommonRoad uses sequences of so called states to describe the movement of traffic participants. Each state consists of a time (aka simulation step), a position, an orientation and a current speed. These properties may be given exactly or in form of an uncertainty interval. CommonRoad does neither care about whether movements are realistic nor possible events occurring between time steps. To cope with this one would need to decrease the time step size and introduce much more states. This would need a very precise prediction of participants. Since these problems CommonRoad alone is not enough for the purposes of my work. Both OpenDRIVE and CommonRoad are not able to specify tests.

Paracosm [18] is a language which tries to specify both environments and tests at the same time. It also allows in a similar way to CommonRoad to define uncertainty intervals for some variables like speed or the number of lanes to define a range of test scenarios simultaneously. Paracosm solves parts of the communication problem mentioned in subsection 1.1 introducing monitors. Monitors allow to read information from the simulation and to define callbacks triggered by certain events during the simulation. Since Paracosm does not explicitly solve the problem of communicating with AIs and is incompatible with the way BeamNG describes scenarios it would be difficult to use it in my work.

The Autoware.AI open source project [8, 14, 15] provides a complete intelligence for autonomous cars which is able to collect sensor data, evaluate the data including localization and prediction of traffic participants and react to the situation. The project does also provide a simulator based on ROSBAG [7] files.
The Apollo project [5] provides an architecture satisfying requirements for developing, testing and deploying autonomous cars. It is already a huge project working together with many companies and providing a comprehensive suite for multiple objectives in the topic of testing autonomous cars.

The approach in [16] describes a unified cloud platform suitable for supporting many autonomous driving applications and even discusses usage of concrete software to solve problems concerning performance, utilization and energy efficiency.

Figure 1: System architecture — This figure shows possible elements in test case definitions, the main components of the cloud and how these are connected. Additionally the relation to the interaction with the tester is visualized.



3 Proposed Methods and Tools

3.1 Proposed strategies

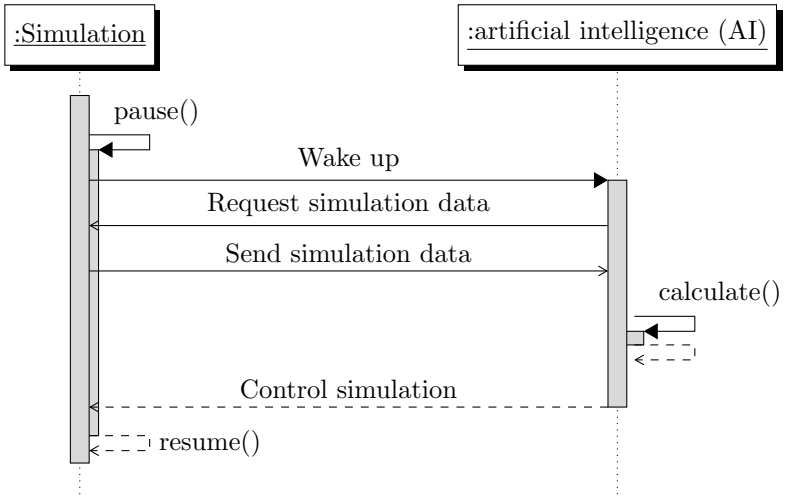
I propose the architecture shown in Figure 1 for DriveCloud. The main parts are the test scenario definition and the cloud itself.
The scenarios follow a modular approach and are represented by two types of files namely Drive Cloud Environment (DCE) files describing simulated environments and Drive Cloud Test Suite (DCT) files containing the actual test cases. This allows reuse of DCEs, avoids duplication and saves bandwidth during transfer.

The scenarios include roads, participants and obstacles which are defined in a similar way to CommonRoad. Roads are collections of lanes where each lane is specified by two sequences of points one representing the left border the other the right one. Participants have an initial position and orientation. Other (static) obstacles have a shape that is a collection of points describing its position, width, length and height. The system is planned to be extensible by allowing to define DCEs not only by DCE files but also by CommonRoad and OpenDRIVE files. To handle these formats they are unified from OpenDRIVE to CommonRoad (based on [4]) to a DCE file and then passed to the generation.

The test cases consist of multiple base sections. One section describes the paths of traffic participants which are sequences of states. Each state contains a waypoint which is a tuple of a position and a tolerance radius allowing a participant to not perfectly reach the position but to go through within a certain radius, a speed definition which either sets a speed the vehicle should have or a speed limit disallowing an AI to be exceeded and an "AI controlled" flag. If the "AI controlled" flag is set to true the simulator request the appropriate AI for controlling the participant.
The preconditions section as well as the success and fail sections contain validation expressions. A validation describes under which conditions the inner conditions have to be evaluated. If a validation condition is not met the inner conditions are not evaluated and do not influence the result of the parent expression. The conditions within validation conditions may be further validation conditions or other conditions like position, speed or damage requirements a given participant has to fulfill. The separation of validation conditions and other conditions allows to restrict whether conditions have to hold at the current tick. This allows to define more expressive conditions e. g. a participant must not exceed a certain speed limit while being within a certain area or on a certain lane. *TODO Find better names for validation conditions and other conditions* Per convention if the preconditions are not satisfied the test is `skipped`. If the success (failure) conditions are triggered the test is `successful` (`failed`). If neither success nor failure is fulfilled the test is considered as `undetermined`. *Mention other states like error and abort?*
DCT files also allow to define named regions which again allows to reuse regions in multiple condition statements and avoid duplication.
The DCTs are passed to the validation module where simple contradictions are checked. *TODO To which extend?* If the validation fails the test is marked as `invalid`.

The generated DCEs and validated DCTs are put together into the simulation and executed. During the execution the simulator needs to communicate with AIs controlling traffic participants. The proposed protocol (See Figure 2) uses four messages for each AI control request. The first call from the simulation wakes the AI up telling it that the simulation is paused and waits

Figure 2: Communication protocol AI ↔ Simulation — This scheme describes the messages sent when exchanging data between an AI and a running simulation



for any control commands. Next the AI requests data provided by the simulation by transferring a list of data required. After receiving this list the simulator sends the appropriate data. The last call sends commands which the AI calculated controlling a participant or the simulation. Having four messages instead of only two messages allows the AI to decide each time triggered which data is currently needed. This allows different AIs to request different data and saves bandwidth of the cloud infrastructure.

Besides the pure technical issues the system will also collect data about simulations, tests, scenarios, results and usage. *Really this information? Something else? Something not?*

3.2 Proposed Tools

TODO Does this section contain duplicated stuff of the introduction?
Describe every tool going to be used? BeamNG, XML parser, Proto buffer, programming languages,...

4 Planned Evaluation

Goals of work:

- *Fast setup of test cases*
- *Expressive conditions*
- *Many concurrent simulations*

- *Automatic evaluation of test results*
- *Collect data about executions*
- *Provide shared computational power*
- *Compatibility with OpenDRIVE and CommonRaod*
- *Help other researchers*

Based on these: What to evaluate?
What could be interesting for someone reading my work?
Ideas:

Check range of variety of scenario descriptions *(dce evaluation)*
Take a big database and check whether these test cases can be generated
***Problem**: Definition of whether a scenario could be generated*

Check expressiveness of conditions *(dct evaluation)*
Check which complexity of conditions can be expressed by conditions.
***Problem**: Measurement of "expressiveness" What is "expressive"?*

Size of configuration file *(Simplicity of dce and dct descriptions)*
Compare file sizes of descriptions in multiple formats.
***Problem**: Capabilities of OpenDRIVE, CommonRoad, Paracosm and DCE are very different. The capabilities of Paracosm could be the nearest. Does file size really matter?*

Easy understanding of descriptions *(Simplicity of dce and dct descriptions)*
Take random computer science related people and ask them whether they can guess what the scenario looks like and what the conditions express.
***Problem**: Is this interesting for anyone reading my work? How to define whether a person could guess that or not?*

Analyze load balancing *(cloud evaluation)*
Take lots of computational power and many simulations and analyze how these are distributed.
***Problem**: Probably this would require quite an amount of computational resources. What is "good"?*

Check elasticity *(cloud evaluation)*
Not interesting/applicable in terms of this work.

Check whether the sytem help *(Usability study)*
Find people willing to test the system and make them to give feedback.
***Problem**: Find people willing to do so. May result in weeks of bug fixes over and over since the software is in pre-alpha without getting results.*

Is an evaluation of any cloud property interesting since there are already tons of papers and improvements to cloud computing strategies also applicable in this case? If cloud properties are interesting quite an amount of computational power and machines needed?
An evaluation of the effectiveness of any property concerning simulation may result in an evaluation of BeamNG not of my work.

5 Schedule

The thesis is limited to a maximum time period of 6 months. Starting at the 8th of April this results in 26 weeks ending on 6th of October. These 26 weeks are divided into tasks and milestones according to Table 1.

Table 1: Thesis Schedule — This table also contains the planned milestones and their names

Weeks	**Task**
1 — 2	Ontology/protocol definitions
3 — 5	Implement generator
6	Implement validation
Milestone M1	"Rocket launch"
7	Setup environment
8 — 9	Implement setup mechanisms
10 — 12	Implement load balancing and scalability
13 — 14	Implement data collection
Milestone M2	"Over the sky"
15	Test system
16 — 20	Evaluate system
Milestone M3	"Nosedive"
21 — 26	Write thesis
Milestone M4	"Final destiny"

After the 26th week there are 2 days left until the available time is fully used. These are planned to print the thesis and hand it in.

6 Success criteria

The system to develop shall satisfy all Must-Have requirements to be considered as successful.

Specification R1 Definition of an ontology for DCEs.

Specification R2 Definition of an ontology for DCTs.

Specification R3a Definition of the message formats used for the communication between simulator and AI.

Specification R3b Definition of the message formats used for the communication between simulator and AI.

Specification R4 Generation of scenarios out of DCE files.

Specification R5 Basic contradiction check for DCT conditions.

Specification R6 Scalable cloud infrastructure.

Specification R7 Collect data of simulations, execution and tests.

The system should implement May-Have criteria to further increase the number of use cases and capabilities provided to users of the system.

Specification M1 Support for OpenDRIVE files as DCEs.

Specification M2 Support for CommonRoad files as DCEs.

Specification M3 Perfect size match of obstacles. In the DCE ontology static obstacles may have an arbitrary complex shape. To generate static obstacles of perfect size the generation needs to use a broader range of objects available in BeamNG and to rotate and position these in a more challenging way.

Specification M4 Use ROSBAG files for logging.

Specification M5 Elasticity support for the cloud infrastructure.

This work will not focus and implement the following aspects.

Specification N1 Real time requirements in simulation and reaction of AIs to the scenario.

Specification N2 Support for various Linux distributions and MacOS.

An overview over all specifications is listed in Table 2.

Is this "summarizing" table worth something?

Table 2: Summary of the Expected Thesis Features

Feature	**Must-Have**	**May-Have**	**Must-Not Have**
Specification R1	×	—	—
Specification R2	×	—	—
Specification R3a	×	—	—
Specification R3b	×	—	—
Specification R4	×	—	—
Specification R5	×	—	—
Specification R6	×	—	—
Specification R7	×	—	—
Specification M1	—	×	—
Specification M2	—	×	—
Specification M3	—	×	—
Specification M4	—	×	—
Specification M5	—	×	—
Specification N1	—	—	×
Specification N2	—	—	×

References

[1] Upamanyu Acharya. What is tickrate, and is it really that important?, July 2016.

[2] M. Althoff, M. Koschi, and S. Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 719–726, June 2017.

[3] M. Althoff and S. Lutz. Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1326–1333, June 2018.

[4] M. Althoff, S. Urban, and M. Koschi. Automatic conversion of road networks from opendrive to lanelets. In *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 157–162, July 2018.

[5] Baidu. Apollo.

[6] C. Erbsmehl. Simulation of real crashes as a method for estimating the potential benefits of advanced safety technologies. *21st International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, August 2015.

[7] Tim Field, Jeremy Leibs, and James Bowman. rosbag - ros wiki.

[8] The Autoware Foundation. Autoware.ai.

[9] BeamNG GmbH. Beamng.

[10] VIRES Simulationstechnologie GmbH. Opendrive.

[11] Tom Gruber. Ontology.

[12] Tom Gruber. What is an ontology?

[13] Nikolas Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. June 2013.

[14] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, November 2015.

[15] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '18, pages 287–296, Piscataway, NJ, USA, 2018. IEEE Press.

[16] S. Liu, J. Tang, C. Wang, Q. Wang, and J. Gaudiot. A unified cloud platform for autonomous driving. *Computer*, 50(12):42–49, December 2017.

[17] D. Loiacono, L. Cardamone, and P. L. Lanzi. Automatic track generation for high-end racing games using evolutionary computation. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):245–259, September 2011.

[18] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. Paracosm: A language and tool for testing autonomous driving systems, February 2019.