Saarland University
Computer Science Department

Master Thesis

Evolutionary Test Generation for Autonomous Vehicles

Author:
Marc Müller, B.Sc.

Supervisor:
Prof. Dr.-Ing. Andreas Zeller
Reviewer:
Dr. Stefan Nürnberger

Advisor:
Alessio Gambi, Ph.D.

Submitted Wednesday 12th September, 2018

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlich wird.

Saarbrücken, 12.09.2018

Marc Müller

Abstract

Autonomous vehicles are becoming an increasingly relevant part of the automotive industry and will only become more important in the near future. Ensuring safety is naturally important when handing over full control of a vehicle to software, but neither industry nor authorities have settled on a standard way to certify autonomous vehicles. Deploying autonomous vehicles for testing in regular urban traffic is a common but costly and risky method. Simulated, or virtual, tests have been introduced as a way to expose problems before deployment, but traditional software testing techniques cannot cope with the massive amount of situations an autonomous vehicle faces. For limited cases, more advanced techniques like search-based testing show more promising results. This work will continue in that direction, utilising procedural content generation and genetic algorithms to evolve driving tasks meant to test the lane keeping capability of autonomous vehicles. Meaningful metrics to characterise input scenarios, output behaviour of the car, and how well tests cover the input space will be introduced to guide test suite evolution towards stressing the vehicle's lane keeping behaviour effectively.

Zusammenfassung

Autonome Fahrzeuge sind ein zunehmend wichtiger Teil der Automobilindustrie und werden in naher Zukunft weiterhin an Relevanz gewinnen. Die Gewährleistung der Sicherheit bei programmgesteuerten Fahrzeugen ist offensichtlich notwendig, jedoch haben sich bisher weder Industrie noch Behörden auf standardisierte Testverfahren geeignet. Autonome Fahrzeuge im Alltagsverkehr einzusetzen ist eine bislang übliche, aber teure und riskante Methode zum Testen. Simulierte oder virtuelle Testverfahren wurden als Mittel vorgeschlagen, Fehler vor dem Einsatz dieser Fahrzeuge zu identifizieren, aber handelsübliche Software-Testverfahren sind der Vielfalt von Situationen, die autonome Fahrzeuge bewältigen müssen nicht gewachsen. Suchbasierte Methoden haben sich hingegen als vielversprechender erwiesen. Meine Arbeit forscht weiter in diese Richtung und setzt prozedurelle Methoden zur Straßengeneration & genetische Algorithmen ein, um die Spurhaltefähgikeiten von autonomen Fahrzeugen zu testen. Metriken zur Analyse von Testszenarien, dem Verhalten des Fahrzeuges und inwiefern Testsätze mögliche Szenarien abdecken werden definiert und angewandt, so dass Tesgenerierung das Fahrzeug sinnvoll und effektiv auf die Probe stellt.

Contents

1 Introduction

Over the past few years, the topic of automating cars with the help of software has become an important part of the industry and will only grow in relevance [1]. The level of automation varies. SAE International has grouped different degrees of automation into five distinct levels [2], starting at Level 0 for no automation, going to Level 1 for advanced driver-assistance systems (ADAS) that help the driver operate a vehicle, and increasing automation up until Level 5 — fully autonomous vehicles, correctly navigating through roads, and obeying traffic laws. Steering wheels are considered *optional* at this level.

There are several ongoing projects progressing towards fully autonomous vehicles (e.g., Waymo [3], Tesla Autopilot [4], Uber [5], etc.) The state of the art in this field is mature enough to be currently under testing in everyday urban traffic [6, 7, 5]. These systems promise to increase safety [8], presuming software to be more reliable at driving than humans. In reality, this is not the case yet. The vehicles — partially or fully automated — deployed in urban environments experienced numerous software faults [9, 10], have caused crashes [11, 12, 13], and even caused fatal accidents [14, 15, 16]. Such faults are apparent even in experimental testing before any deployment [17, 18]. Especially cases where the cause of the accident is determined to be a software fault alone — such as the fatal Uber crash where the autonomous vehicle's AI failed to identify a pedestrian [5] — motivate more rigorous testing of autonomous vehicle software.

The importance of such testing is evident from the severity of their failures and also makes sense intuitively: Failures in safety-critical software that's meant to operate vehicles with human passengers can easily lead to crashes. In the worst case, these would be deadly. Thus, before fully autonomous vehicles or even just partially automated systems can enter the mainstream, their functionality needs to be tested extensively. This includes the aforementioned safety concerns, but also the quality of driving [19]. A car that might be able to travel from point A to point B without an accident could still be uncomfortable for the user simply due to the car driving poorly (e.g., sharp turns, sudden de-/acceleration, causing motion sickness, etc.)

Despite the importance of testing, the industry has yet to settle in on some standardised procedures to test automated vehicles [20]. Existing methods to ensure the safety of non-autonomous vehicles before deployment don't translate well into the space of autonomous cars, as others have argued [21, 22, 23]. How to apply traditional software testing techniques to autonomous vehicles is also a fairly open topic, as shown in Section 3. Putting autonomous vehicles on trial in the real world or tailored driving challenges is a common approach [24, 25, 26, 6]. These are called "Naturalistic Field Operational Tests" (N-FOT.) Trials like that would need to cover hundreds of billions of kilometres driven to reliably compare their failure rates to that of human-driven vehicles [27].

An alternative to Naturalistic Field Operational Tests are simulated tests. To avoid the cost of setting up a physical testing environment or the risks associated with testing autonomous vehicles in normal traffic, the software controlling an autonomous vehicle is fed with simulated data to see how it responds [28]. This is becoming a more common practice, a shown by Uber suspending its N-FOT testing programme in favour of simulation tests due to a fatal accident [15, 5]. Vehicle testing in simulation varies in scope, some works focusing on simulating a single or few sensors to test specific functions of the vehicle such as speed & position sensors [29, 30], testing motion planning [31, 32], going for a realistic simulation of visual data to test lane and object detection capabilities [33, 34, 35, 36, 18, 17], or evaluating behaviour at a higher level of abstraction to see how the vehicle acts in traffic or other scenarios [37, 38, 39, 40, 41, 42, 43]. Section 3 goes into the works that have shown simulation testing to be effective for autonomous vehicles.

Depending on the type of simulation, requirements for the simulation framework can be very stringent, as the simulator needs to provide data such as realistic graphics, run complex physics computations to accurately model movements that are difficult to generate efficiently, and model driving scenarios to be executed in the simulation. Given the multitude of factors that impact driving in the real world, for a simulation to have external validity, it needs to include many of these factors. This inherently makes the task of providing simulations hard. Some of these problems have been worked on in the context of driving simulators for humans [44, 45, 46], but also video game engines, which often take on the task of simulating 3D worlds with physics in real time. Both of these are already established in research, with video game engines in particular providing simulations good enough to have already been used in testing of autonomous vehicles [47] and training deep neural networks which were then able to drive in the real world [48, 49, 50]. This means, not only were the simulations suitable for testing, they even successfully served to provide ground truth data for an AI to learn how to navigate in the real world.

With the use of simulators there also comes the challenge of defining the content of the simulation; the layout of the roads, the environment roads are in, obstacles, other vehicles, and so on. These are things that either naturally exist in the real world, or were built over years of civilisation, but need to be re-created virtually to be used in a simulator. The aforementioned work by Chen et al. [48] and Johnson-Roberson et al. [49] relied on content that the simulator or video game shipped with. For example, [49] relied on the rich open world found in Grand Theft Auto V [51], which features varying weather conditions, time of day changes, other vehicles on the road, pedestrians, and so on. That content was hand-crafted by over 1000 people over five years [52]. Creating more or changing existing content would require a non-trivial amount of work by the testers [53]. This motivates automating generation as much as possible.

The problem of such content creation affects the gaming industry just as it affects the use of game engines as simulators. Because of this, the field of procedural content generation (PCG) has emerged within the gaming industry [54, 55]. PCG tries to solve the problem by algorithmically — as opposed to manually — producing content for games. With regards to driving in particular, works like [56] have created methods on how to produce city-like street networks and [57] describes a method of creating and refining race tracks through evolutionary computation. These works focus on making content engaging to humans and need to be adapted to be useful for the domain of testing autonomous vehicles.

Naturally, generative approaches can also be applied for other purposes. In [58, 59, 60, 61] the authors automatically generate content to test components of ADAS or autonomous vehicles. In this context, the content generation is comparable with automatic test generation, along with other concepts like search-based procedural content generation [62] aligning with search-based test generation. As argued in [58, 60], the amount of options to generate tests is huge, even restricted to the domains of the respective papers (pedestrian detection and navigation.) A way to identify useful tests and guide generation accordingly is needed to be able to conquer the sheer amount of options.

One such way are genetic algorithms (GA) [63]. GAs are an optimisation method that relies on random combinations and mutations of a population according to some fitness for a purpose. In

[58], for example, this was used to rank scenarios of pedestrians crossing a street according to how close they were to a vehicle, looking for tests that would likely cause a collision. This way, the testers were able to guide generation towards relevant test cases, saving time on redundant and uninteresting ones.

With regards to lane keeping, Tian et al. [17] used search-based testing modify images of streets to simulate different weather conditions. Using images taken from real roads they generated images of those roads under varying weather and lighting conditions and successfully exposed inconsistent behaviour in the vehicle AI under test compared to the reference set. The predicted steering angle of the AI for an input image served as the basis to detect inconsistencies: If the AI predicated a different steering angle after modifying the weather, it was considered inconsistent. No driving was performed — simulated or real.

1.1 Problem Statement

Autonomous vehicles should be able to correctly drive down a lane. Staying within a lane is the one of the most fundamental tasks in urban driving. However, state of the art AIs can be fooled into inconsistent lane keeping by varying weather conditions [18, 17] and lane detection methods in general aren't fully reliable yet [64]. Even among leaders of the industry, this task is not solved, as shown by accidents like when Tesla's Autopilot ended up steering into a concrete lane divider [12, 16]. One of these crashes was fatal. In these accidents, the vehicle collided with the environment. Failing to keep to the lane can also result in the vehicle driving into oncoming traffic, which is a particularly dangerous situation for everyone involved. As such, testing lane keeping behaviour in N-FOT studies carries many risks with it and has already cost lives. Simulation tests are a way to avoid these risks, but, as Section 3 shows, current research is lacking methods of content creation to test lane keeping in simulation. State of the art research has only focused on steering angle prediction [17], not actual lane keeping, or employed naïve approaches like random generation that leave doubts as to how effectively the search-space was covered [59]. Within this domain, the amount of options during testing is massive. Enumerating them would encompass every lane a vehicle could plausibly have to drive on. Depending on which aspects of real-life lanes are considered during testing, this could include the Cartesian product of every possible length of road, shape of a turn, width of lanes, variations in width, amount of lanes in a road, in-/decline of roads, random artifacts in the ground like potholes, and so on. If one expanded into how a road looks visually — since visual processing is a common component in autonomous vehicles [65] — the amount of options grows even more: Every possible lane divider, lane divider length, colour of dividers, ground texture, environmental factors such as weather & time of day, and so on — the possibility space is huge. This makes it unlikely that random search covers a meaningful subset of it and a more sophisticated method is required.

In this thesis I take on the problem of automatically generating effective tests for the lane keeping ability of autonomous vehicles. These tests require the vehicle to drive along a certain path through complex road networks. To this end, I propose an evolutionary approach to road network generation that iteratively refines a test suite towards detecting erroneous lane keeping behaviour of autonomous vehicles. I introduce algorithms for road generation and operations that enable the use these networks a genetic algorithm. For these I define formalisations of road networks that leverage prior work on encoding road geometry (e.g., CommonRoad [32], OpenStreetMap [66].) All of these components are required to produce valid and solvable tests for the ego-vehicle to solve. How I achieve this is described in detail. To make this task more manageable, the search space is restricted to fixed-width flat roads with a lane on each side, but ideas to expand this in future work are presented.

How to analyse a vehicle's performance regarding lane keeping whilst driving on these roads is defined and how that factors into the evolution of a test suite is discussed. Generated tests are executed within a driving simulator, expanding the current body of work regarding automatic test generation for simulation testing of autonomous vehicles. As per [17], the diversity of input in testing steering angle predictions plays an important role. How to achieve higher diversity with an approach like this is shown and evaluated.

The method I propose is deliberately kept generic enough to be able to work with various driving simulators, but the considerations that went into designing tests intended for simulation and which factors go into deciding on driving simulators are presented. The simulator used during the evaluation is BeamNG.research [67], a simulator with particular focus on driving physics, which makes it suitable for evaluation of driving behaviour.

I evaluate the proficiency of my approach with regards to eliciting lane keeping failures and producing diverse test suites in a series of experiments that gradually refine the generation process from a simple but effective one that focuses on one road at a time to a more general and improved approach able to generate convoluted networks of multiple roads. The first batch of experiments focuses on evolving tests with single roads and comparing them to random generation, showing my approach to be superior and establishing that this domain requires non-trivial generation methods. The second set of experiments improves upon this by expanding test generation to include multiple roads and intersections, demonstrating how inclusion of intersections can reveal more failures than single roads and improve efficiency of the generation. The third and fourth series of experiments will alter fitness evaluation and generation method to foster more diversity in test suites.

My work is proven to be an effective way of finding lane keeping failures within autonomous vehicle AIs in simulation testing and establishes a method more sophisticated than mere random generation. Possible improvements to this are presented for future work.

1.2 Contributions

The main contributions of my work are:

- Providing a way to specify roads that's conducive to generation and manipulation.
- Defining operations used to refine such roads towards producing lane keeping failures.
- Showing lane centre distance as fitness is viable to evolve test suites towards producing lane keeping failures.
- Showing trivial approaches like random generation are insufficient in this domain.

1.3 Structure

The thesis is laid out in the following sections:

- Section 2 provides background information on the topics of road specification, genetic algorithms, and video game engines.
- Section 3 critically discusses the current state of the art of relevant research, taking a look at autonomous vehicle testing, simulation tests, and procedural content generation in particular.
- Section 4 formalises the concepts used in road generation, describes the algorithms and operators used in generation & manipulation, how tests are executed and evaluated.
- Section 5 gives an overview on how the presented method is implemented.
- Section 6 describes the empirical evaluation of my method. It covers a series of experiments carried out to examine the merits of my approach, presents their results, and discusses their implications for my and future work.
- Section 7 provides an overall discussion of the work and what conclusions can be drawn from it.
- Section 8 contains avenues for future work that would improve the approach and evaluation.

2 Background

This section provides necessary background information required to understand the thesis, included to make the work more self-contained.

2.1 Genetic Algorithms

A genetic algorithm (GA) is an optimisation technique inspired by the theory of evolution, which iterately refines a population of individuals according to some fitness [63]. What “population”, “individual” and “fitness” means is specific to the domain a GA is applied to. For example, in my case, the population is a test suite, each individual a test, and fitness is the test’s effectiveness in producing failures. From these individuals, the GA selects pairs according to their fitness and applies so-called crossover and mutation operations to them to produce offspring. What these operators do is also domain-specific. In my case, for example, crossing over two roads means splitting them at random points and criss-cross combining the resulting parts. Mutating is, for example, altering a turn to be sharper or wider. With mates selected and offspring produced, the GA replaces the current population with the offspring one and repeats the process for this new generation. Over time, randomly selecting more fit individuals and re-combining & altering them leads to suites of individuals with high fitness.

It's important to note that selection does not strictly focus on more fit tests. It is intentionally randomised to enable more exploration of the search space. A common method of selection is called Tournament Selection, where individuals are picked randomly to enter into metaphorical tournaments where the most fit individuals win and get selected. This is random, because the entrants to the tournament are selected randomly, but gives preference to fit ones as, the winner is determined by fitness.

To guarantee some fit individuals *do* survive until the next generation, a GA can be configured to use what's called "elitism." With an elitism of one, for example, a GA will move the most fit individual unaltered into the next generation. For an elitism of two, it would be the two most fit individuals, and so on. This avoids killing off very fit individuals due to mere randomness.

2.2 Polylines

When representing roads digitally, there is no obvious choice of format. The shapes roads can have in real life are very complicated. Depending on the domain, different representations are more suitable. For example, in visual processing, road representations should naturally include visual artefacts of roads such as the colour and shape of lane markers or the texture of the ground. In motion planning, these details are less important and the shape of the road is given more focus. My work is in this domain and a look at related work reveals so-called polylines to have been used effectively in projects like CommonRoad [32], OpenStreetMap [66], and OpenDrive [68]. Polylines are a way of defining lines similar to how polygons are defined: as a discrete sequence of points [69]. In the context of roads, they have been used for encoding by specifying polylines that describe the outer edges of roads/lanes. Their utility in this domain is the ease with which precise shapes can be defined. Within my work, I consider a polyline as:

Definition 2.1 (D2.1)**.** A polyline $p = ((x_1, y_1), ..., (x_n, y_n))$ is a finite sequence of two-dimensional coordinate pairs — also called "vertices."

This definition can be expanded to arbitrary dimensions, but in my work, only two-dimensional lines are needed because roads are assumed to be flat, so this definition suffices.

Certain operations are computationally easy with polylines, these include:

- Testing for intersections by looking for adjacent pairs whose corresponding line segments intersect. This is important in my work, as it can be used to detect intersections between roads.
- Computing the length of a polyline by adding up the lengths of each adjacent vertex pair's respective line segments. For road networks, this can be used to measure the length of a road and the distance driven along it.
- Calculating the shortest distance of a point to the polyline by finding the shortest distance to any adjacent vertex pair's respective line segment. For example, given a lane and the line describing its centre, the distance to said centre can be calculated easily. When done with the position of the car, this value expresses how far away the vehicle is from the lane centre.
- Applying affine transformations by applying them to each vertex. This allows for manipulation of roads using common geometric operations such as rotation and translation.

2.3 Video Game Engines

Video games are technologically quite complex. Modern games usually include a variety of computationally expensive tasks like rendering of high-resolution 3D models including textures, playback of locational audio and music, simulating physics, real-time lighting simulation, dynamic weather, dynamic time of day, artificial intelligence for non-player characters, networking for multiplayer, efficient I/O operations to enable smooth simulations, and, in some cases, even automatic generation of content to play. These components have the additional requirement of being computable in real time, as video games ideally rely on refresh rates above 30Hz [70]. Reimplementing and optimising such a vast set of features for each game would obviously be wasteful. Instead, as common software engineering practice recommends, these components get re-used for various games, as shown in all the games using common engines like Unreal Engine [71] or Unity 3D [72]. The bundle of these features and the API they are exposed through is what's called an “engine”. It is similar in concept to a framework or software development kit.

Many of the above features are relevant to the domain of driving simulations. Recreating vehicles for simulations can be done with 3D models and textures, driving relies on simulation of physics, producing sensor data such as a camera feed can be achieved by rendering the game world from the camera's perspective, and so on. My work in particular relies on BeamNG.research [67] as the simulator, which is based on the Torque3D [73] engine and expands it to include detailled soft-body physics simulations. Because of the overlap in problems games and simulators have to solve, game engines have shown to be a useful tool in vehicle testing, as discussed in Section 3.

3 State of the Art

The fields of procedural content generation and autonomous vehicle testing are by now fairly large. What follows is a closer look at current research indicating the state of the art.

3.1 Naturalistic Field Operational Tests

Autonomous vehicle testing is still a fairly open field [20] where a common approach is to simply test the vehicle in the real world. These so-called “Naturalistic Field Operational Tests” (N-FOT) produce the most reliable and accurate results, as the car is being tested in the way it's supposed to be used: Driving itself around. The ego-vehicle is deployed with the sensors, actuators, and the software required for operation and, optionally, additional testing software to aid evaluation of the vehicle's performance. Such a system is described by Belbachir et al. [74], where a vehicle equipped with four cameras, a GPS, a LIDAR, and an intertia sensor was instrumented with hardware actuators (seen in Figure 1) including software meant to evaluate the obstacle detection and path planning functionality of a vehicle as it is being driven. Similarly, Althoff et al. [75] implemented an online verification method for autonomous vehicle testing. The test subject was equipped with software that verified the vehicle's decision making by performing reachability analysis at runtime and ensuring the vehicle's projected path never collides with surrounding objects. While being the most authentic way of testing, N-FOT studies are impractical and, in case of vehicle faults, even dangerous. There have already been fatal crashes involving

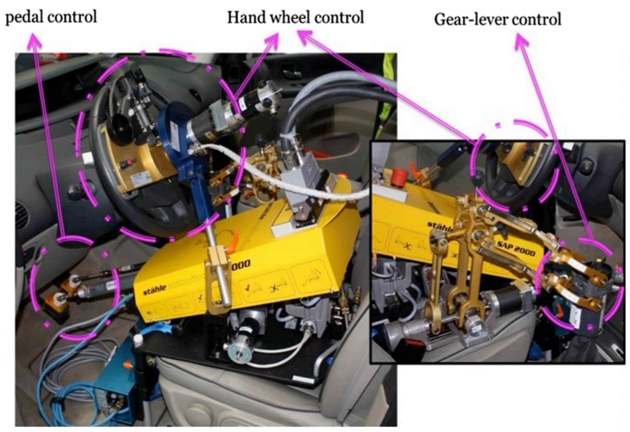


Figure 1: The hardware a vehicle is equipped with by Belbachir et al. [74] for online testing.

autonomous vehicles or advanced driver assistance systems in testing [76, 16, 15]. Besides the danger, it's also less practical to deploy a testing vehicle for every run to gather data. This impracticality has implications for the certainty of the results: for a reliable safety comparison between autonomous and human-driven cars, it would be required to perform these tests for billions of kilometres to reasonably argue about the safety of these autonomous vehicles compared to human-driven ones [27]. This is because there are so many samples for human-driven cars that any prediction of their risks is fairly reliable. This is not the case for autonomous vehicles. With the technology only being available to few people, they'll have a hard time catching up with the data on traditional vehicles.

N-FOT studies suffer from a bias in where testing is performed, too. An example of this is Volvo's autonomous vehicle project not being able to recognise kangaroos [10], because they never came up during testing in Sweden. For autonomous vehicles to become mainstream, they need to be able to handle a large variety of scenarios. Currently, however, testing is slanted towards locales where autonomous vehicles are being developed and the conditions specific to those environments.

3.2 Simulation Testing

Simulations have emerged as a way to deal with some of the problems of N-FOT testing. These don't have to be fully simulated tests, but can be simulations based on data from N-FOT studies like Zhou et al. [77]. They used a vehicle's GPS information while driving to re-create the roads it has driven on. This data was then used to do simulated tests of a vehicle's adaptive cruise control. Zhao et al. [78, 79] used data from N-FOT studies to build a model of driving behaviour. This model then served as reference to run simulations of car-following and cut-in scenarios and to evaluate the safety of an autonomous vehicle. It was possible to dramatically reduce evaluation time by factors between 300 and 100,000 times. These works show promising results, but still suffer from the reliance on initial N-FOT data for model building. This carries with it the aforementioned difficulties and biases in obtaining such data.

A similar idea was described by Müller et al. [80] and Zhang et al. [18]. They took real life camera data as a base and manipulated the recorded images to simulate the same image with different weather conditions. These images were then used to evaluate image processing components used in driving. If a test subject changed its behaviour from the reference images after their weather was varied, it was considered inconsistent. This helped reveal flaws in the image processing algorithm, as it was not robust against different weather situations. An interesting aspect of generating such data was shown by Tian et al. [17]. Their work is similar to Zhang et al. [18]'s in that they manipulated images to find inconsistencies in neural network models used in autonomous vehicles, but they adapted the concept of coverage to the domain of neural networks. During testing, the amount of neurons activated by an input was recorded and test data was selected to maximise neuron coverage. They found that increasing coverage also helped find more flaws. These works address the problem of obtaining data, because, instead of basing their simulations on data obtained from N-FOT studies like the one done by Zhou et al. [77], they could rely on publicly available images thanks to the focus being on image processing. For example, Zhang et al. [18] obtained driving footage under different weather conditions from YouTube.

Completely simulated testing has gained relevance as a way to avoid the drawbacks of N-FOT testing and real world data collection entirely. This method is already used within the industry [81, 5, 82], but the corresponding simulations are highly tailored to the respective product and not publicly available. This area varies in how holistic the simulation is. Tools can focus on simulating specific types of scenarios and sensor data, or try to cover multiple domains to allow testing interactions between multiple components [47]. A testing tool like that is described by Schuldt et al. [42]. It's presented as a construction kit for virtual testing in which the tester can define the testing environment, weather situation, time of day, surrounding traffic, driving task, and so on to then use as tests. It then allows the user to specify which data is needed for evaluation and what criteria decide success or failure. The authors give an example application of the tool where a vehicle assists the driver in navigating a narrow road. These tests were run with slightly varied parameters and the vehicle failed half of them. However, the biggest drawback here has been that the base test needed to be specified manually. From the road to drive on — which the authors first had to scan from the real world with a physical vehicle equipped with the sensors required to do so — to the description of the scenario and how it is supposed to be evaluated, it was a manual process. This is more labour-intensive but also has a bias on behalf of the testers inherent to it, which could prevent faults from being discovered simply because no one thought of the situation that would bring them about.

With the difficulties in test creation, some authors have worked on collections of tests that are supposed to cover important aspects of certain domains. For the problem of identifying drivable space, Fritsch et al. [64] have collected a set of images of roads together with a ground truth of the roads seen in them. This allows developers to test their algorithms against a standardised dataset without having to find their own reference data. They can also compare their performance in this dataset to other approaches. This idea of defining a set of reference problems for testing and comparison has also been applied to the domain of motion planning. CommonRoad [32] defines both a way of specifying motion planning problems for an autonomous vehicle, but also provides a set of problems with optimal solutions motion planning algorithms can be tested against. While such datasets serve as a baseline for passing and comparison, they still require manual curation of data and ground truths for a domain.

An alternative to manual creation is automatic test generation. Instead of the tester handcrafting tests, they are generated through some algorithm. This makes test creation more efficient, of course, but in turn requires an algorithm that's good at creating tests. Depending on the scope of tests, this is far from a trivial task, especially for some of the elements relevant to autonomous vehicles. Generating cities and their road networks, traffic, pedestrians, placing obstacles, and so on, are hard tasks. The focus of generation methods is therefore usually more narrow. Kendall et al. [59], for example, used random generation of road flat road strips to train and evaluate a reinforcement learning model for autonomous driving. Their work varied road shape, texture, and lane markings to successfully train lane keeping in their autonomous vehicles.

A more elaborate test generation method is shown by Abdessalem et al. [58]. They simulate radar data of a pedestrian crossing the road and use that to test autonomous vehicle software. This particular scenario also serves as an example of why N-FOT studies aren't always applicable, because testing a vehicle by having people walk in front of it is hardly ethical. It also addresses two key limitations in automatic test generation: How to reduce the very large space of possible scenarios to meaningful ones and how to deal with the high computational cost of simulations. For the first problem, they employed search-based testing in form of a genetic algorithm that optimises towards "interesting" test cases. "Interesting" in this context meant how close a pedestrian is to the pedestrian detector's warning range, how close they are to the car, and estimated time to collision. The lower these are, the more interesting the test. To cut down execution time they relied on surrogate testing using a machine learning model that was trained to predict likely test outcomes and helped avoid some redundant executions and aided search. Their results showed they were outperforming a trivial approach like random search, both in speed and effectiveness.

Regarding the search for effective tests, Mullins et al. [60] looked into and adaptive generation method that identifies "performance boundaries" of the vehicle and helps to filter out redundant tests that lie within the same performance category. The task the vehicle was given was driving from one point to another while avoiding randomly placed obstacles. These tests were executed in simulation. A "performance boundary" in this setting was determined by examining where small changes of the input scenario cause large changes in the vehicle's behaviour. The intuition was that similar behaviours of the subject need not be tested multiple times and testing should instead focus on the inputs where behaviour is different. With this, they were able to find input boundaries that, for example, separate tests where the vehicle's decision-making fails to lead it to the goal point from ones where the task is completed successfully.

As it stands, simulation testing is emerging as a field to overcome difficulties in N-FOT studies, but is lacking in terms of test creation.

3.3 Video Game Engines as Simulators

With simulation testing, there's the implicit need of a simulator. For the domain of driving and the elements involved therein, the requirements for such a simulator are quite high. Depending on the task at hand, it needs to simulate pedestrians, traffic, varying weather, time of day, provide certain sensor data (GPS, Lidar, etc.), vehicle physics, and so on. To an extent, these are problems the video game industry has also faced. Rich open world games like Grand Theft Auto V (GTAV) [51] feature large cityscapes with changing weather & daytime, numerous pedestrians, and AI-controlled vehicles in engaged in traffic.

The viability of such engines for image processing has already been established [83], but regarding autonomous vehicles in particular, they have also served as generators for ground truth data in

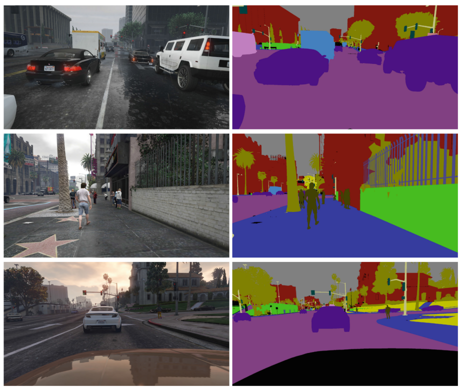


Figure 2: Example images of the pixel-wise object annotation based on GTAV used in [85].

machine learning. Chen et al. [48] used TORCS, an open source racing simulator [84], for deep learning with regards to lane keeping and vehicle detection. In the area of lane keeping, the trained network was successful when used to predict steering angles and detect lanes in real life footage. For vehicle detection, however, the model was less reliable. In line with intuition as Nentwig et al. [83] argues, the authors attributed this to the lack of realism in how TORCS displays cars.

Training neural nets with data from more realistic simulators has shown more promising results. For example, the aforementioned GTAV was used in both [49] and [85] for machine learning of object and lane recognition. To achieve this, they modified the game to provide pixel-wise annotated frames that paint each object with a pre-defined colour which serves as the ground truth for recognition (seen in Figure 2.) After training, both works compared their models using the KITTI benchmark [64] and found their approach to be as reliable as prior work with real images. Despite this success, [53] goes into some limitations of these approaches. Most notably, the fact that GTAV was not developed as a research product and is not open source. To create a specific scenario, the user needs to rely on third party hacks and modifications. This is a cumbersome process with only limited possibilities, because the game is only modifiable to a certain extent. Their work instead recreated a scene of a pedestrian crossing and a specific vehicle crash — the fatal Tesla crash from 2016 [76] — in a more customisable engine, Unity 3D [86]. They were able to rely on real-world map data from Mapzen [87] to recreate that scene, which would not have been possible in GTAV without considerable effort. Similarly, [59] used an openly available engine, Unreal Engine 4 [88] to implement their simulated test generation discussed earlier.

Unity 3D was also used in [61] to synthesise image data of pedestrians in strange situations where state-of-the-art object detectors are likely to fail. This was used to make up for lack of data on rare, but important, occurrences such as a person climbing a fence that should still be recognised. The authors were able to produce a data set that increases the accuracy of pedestrian detection in state-of-the-art detectors.

Several projects are currently being developed that use game engines and adapt them for autonomous vehicle testing. Based on Unreal Engine, there exist CARLA [50], AirSim [89], and

DeepDrive.io [90]. Similarly, Udacity has built a simulator for use in their autonomous vehicle education programme based on Unity 3D [91]. The simulator used in this thesis is also a variant of the video game BeamNG.drive [92] specifically tailored towards research in this area, BeamNG.research [67].

To summarise, game engines as simulators are established for ground truth extraction [48, 49, 85], but, because video games are usually released as entertainment products with authored content, scenario/test creation is lacking. Early work is done to overcome this hurdle in more open engines [53, 59, 48]

3.4 Procedural Content Generation

Creating content for video games takes a lot of work [52] and is more of a manual than an automatic process. To make it easier, research has been conducted that tries to procedurally create the content required for video games. This field touches on a variety of domains [93], from generating textures for rendering to entire virtual worlds [94, 95]. Given the focus of this thesis on road generation, this section will cover the state of the art of road generation. More specifically, the focus is on high resolution generation of road geometry. There exist works that produce life-like urban road networks [96, 95, 97], but the generated models are usually low in resolution. Namely, they focus on a macroscopic generation of city networks that does not capture details needed to evaluate lane keeping — they often don't even differentiate between lane and road.

A lot of the work relevant has been in the generation of tracks for racing games. In my case, a vehicle is supposed to drive through a road network without breaking out of bounds. In a racing game, the player is supposed to drive along a track as fast as they can. The requirement to stay within bounds is implied by the slowdown/penalty a player would incur if they left the drivable road. One such generation method can be seen in Loiacono et al. [57]'s work. They encoded race tracks as a Bézier curve and used a genetic algorithm to evolve racing tracks. The genetic algorithm used the defining points of the Bézier curve for crossovers and mutation and measured the curvature and speed profiles of the track to evaluate fitness. The higher the diversity of curves and attainable speeds, the more fit a track was considered. When humans were asked to drive on these evolved tracks, they found the generated tracks to be aesthetically appealing, but only about half of the participants actually preferred the evolved tracks to the manual control one. This very idea was later adapted to use human preference as feedback for evolution [98]. The algorithm generated tracks that were available for download on a website which also allowed rating the tracks. The ratings of the tracks were then used as fitness values during evolution. This has resulted in tests that were appealing enough to be officially included in the racing game they used — TORCS [84].

Georgiou et al. [99] used a genetic algorithm with the player of the game in the loop to refine tracks according to the player's skill level. The algorithm generated tracks as Bézier curves, similar to Loiacono et al. [100], and, for each track, also computed a reference line the player should ideally follow. The player's performance on the track was then evaluated using this line as a reference while they were playing. This information was combined with various data about the player as they are playing, including information from an eye tracker that provided the system with data on what the player is focused on. This allowed the genetic algorithm to refine tracks to match a user's skill level. This was confirmed in experiments which showed the genetic algorithm to produce tracks that cater to players of different skill levels according to their performance in the game.

To summarise, there exist procedural content generation methods that focus on the creation of challenging road shapes — race tracks. These can produce content that is engaging to humans, even focusing on their particular performance. However, they focus on producing content entertaining to humans, not challenging to autonomous vehicles. My work will adapt these concepts to the domain of test generation for autonomous vehicles.

3.5 Conclusion

Simulated tests for autonomous vehicles have emerged as a common practice to avoid danger and cost of N-FOT studies. Video game engines, thanks to dealing with similar problems a driving simulator needs to solve, are viable simulators for such tests. Generating tests for such simulators is then comparable to the field of procedural content generation, in which methods to generate roads already exist. My work advances these fields by adapting procedural content generation methods for automatic test generation for use in game engines as driving simulators. Namely, tests will be generated to provoke the autonomous vehicle fail to stick to the lane. To this end, they will get generated and refined using search-based methods — genetic algorithms in particular — with the vehicle's performance as the fitness, focusing search on tests that make the vehicle drive poorly.

4 Method

My work focuses on fixed-width flat roads. These roads have exactly one lane on each side and can overlap to form intersections. This is only a subset of all possible ways roads can vary, but, as Figure reffig:examplenetworks shows, the road networks possible within these parameters can be quite complex. To generate such networks, I first formalise representations of the individual components road networks are made of, then devise algorithms to generate and manipulate networks in these representations, and finally utilise a genetic algorithm to search for meaningful tests. In this context, "meaningful" refers to tests that get the vehicle to be tested — the so-called "ego-vehicle" — close to or fully breaking out of lane bounds. To this end, I define a fitness function that guides the GA during search towards tests during which the ego-vehicle has high distances from the lane centre. To produce offspring based on these tests, new operators for crossing over and mutating tests are introduced.

Besides the goal of producing lane keeping failures in the ego-vehicle, the diversity of the test suite is considered important. This is motivated by the intuition that confronting the ego-vehicle with a diverse set of situations helps expose unique faults or certify correct behaviour. This intuition is supported by prior work on steering angle prediction [18] which found the diversity of the input set to be correlated with the amount of failures exposed. How tests/test suites can be analysed regarding their diversity is part of my work.

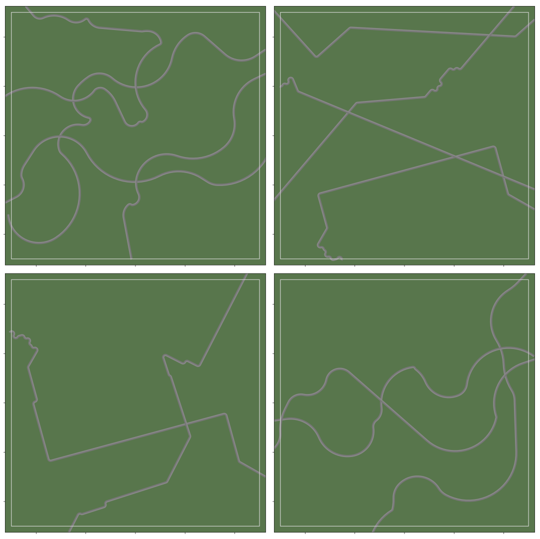


Figure 3: A collection of road networks possible in my representation.

4.1 Approach Overview

As discussed in Section 2, genetic algorithms are an effective way to deal with optimisations in large search spaces. In my approach, the genetic algorithm — from here on called GA — treats tests as the individuals and test suites as the population. Crossover and mutation operators vary the shape of the road networks tests are conducted in. Tests are considered more fit the further away they get the ego-vehicle from the centre of the lane it is driving on. For each generation, the GA evaluates every test's fitness by executing it in a simulator and then randomly combines and mutates tests to create a new generation, giving preference to more fit tests during selection. The intuition here is that picking tests which caused the ego-vehicle further away from the lane centre as parents for the next generation will, over time, lead to a suite that makes the vehicle break out of the lane entirely and, past that, cause more and more such failures.

Overall, the GA follows these steps:

1. Initialise test suite with n randomly generated tests. How these are generated is defined in Section 4.3.3.
2. Repeat until finished:
    (a) Execute tests in simulator to obtain a trace of the ego-vehicle's position while performing the driving task.
    (b) Use trace data to compute each test's fitness by measuring the lane distance of the ego-vehicle during testing.
    (c) Select tests that serve as parents for the next generation using Tournament Selectio.n

(d) Cross over pairs of mates and possibly mutate offspring until either enough offspring has been produced for a new suite of size n or all possible mating pairs have been crossed over.

The above loop finishes when the GA is out of budget. The "budget" is a resource the GA uses up during testing. One such example is time; one could allot a GA twelve hours of processing time and evaluate how much it achieves in that time. In my thesis, I considered generations to be the budget. Because execution in the simulator can be parallelised completely, the computation time measured in real time can vary a lot based on hardware. Generations are a more stable measurement, that's why it was chosen as the budget throughout this thesis.

For the crossover and mutation step it should be noted that the operators used can produce invalid tests. What's considered valid and invalid is described in the formalisations. For now it is important to note that, if crossing over and mutating all possible pairs of mates fails to produce n valid tests for the next generation, tests from the current generation are carried over, leading to a dynamic level of elitism.

4.2 Test Overview

To properly describe how the GA generates, crosses over, and mutates tests, a more detailed explanation of what tests, road networks, and roads actually are in my representation is required. The following sections serve as formal descriptions of that, laying out what tests are made of and what constraints are imposed on them.

A test is the task of driving through a road network from a starting point to a goal point. Road networks need to fit within a fixed boundary to make test execution finite. The target path through this network is fixed as part of the test. This makes analysis of what a test actually required easier, as the path to take isn't up to the ego-vehicle's AI and the GA can evaluate the ego-vehicle's performance along said path. An illustration of a test with the respective components highlighted can be seen in Figure 4. To ensure a vehicle can actually remain within a lane, the road networks used in testing need to have gapless drive-able space. If there was open space along the roads, the test would essentially require the vehicle to leave the lane, making it automatically fail in that event. Therefore, networks are defined and generated such that the drive-able ground meets this criterion and the ego-vehicle always has a chance to reach the goal without leaving the lane. Failure to do so is always due to the ego-vehicle.

Because tests need to be generated and transformed easily, this approach builds upon previous work in specifying shapes of roads such as OpenStreetMap [66] and the definition of driving tasks like CommonRoad [101]. Road networks are encoded as polylines (see Section 2 for an overview on polylines.) In that representation, roads are specified as a series of lines and altering them is performed by applying geometrical transformations to the underlying vertices. With regards to defining the driving task, CommonRoad [101] provides examples on how to do so using polylines. In that work, the authors define the task as a planning problem whose solution is a polyline the vehicle should follow through a road network. In my representation, polylines are similarly used to define the shape of the road, but, since there is only one goal, a test does not require formulating a planning problem. The polyline to drive along is easily computed from the path of the test.

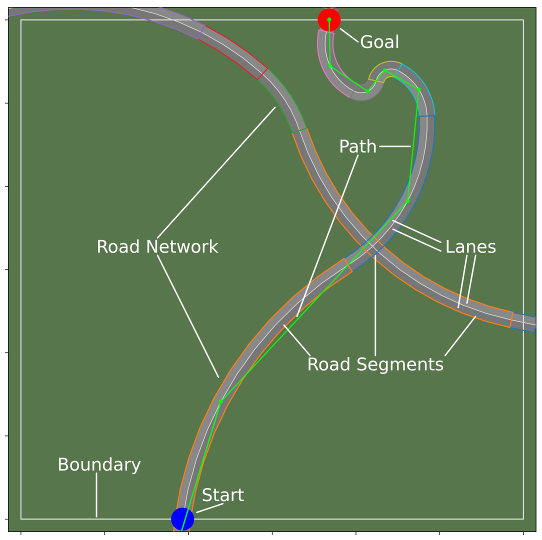


Figure 4: An example test generated with my approach with components of the test highlighted.

Broadly speaking, in my approach:

- A test is a tuple $T = (N, B, P)$, where N is a road network, B is a boundary polygon the road network needs to fit in, and P is the path through N the ego-vehicle has to follow. It has to be traversable without breaking outside of the lane. The start and goal points S, G are where the the first and last segments in P cross the boundary.
- The road network $N = (R_1, ..., R_n)$ is a collection of roads that possibly overlap to form intersections.
- A road $R_i = (S_1, ..., S_n)$ is a sequence of *road segments*. These segments are the building blocks roads are constructed with. For example, a road that is straight for ten metres, then has a left turn, followed by another straight segment of five metres, followed by a right turn can be understood as being composed of four segments: A straight, a left turn, another straight, and a right turn.
- A road segment $S = (L_1, ..., L_n)$ is a series of lanes. Like roads in real life, each piece of road is split into lanes, which are further distinguished by which side of the road they're on. My thesis works on roads with one lane on each side, so this definition can be assumed to always be for $n = 2$.
- A lane $L = (l, r)$ is a pair of polylines that define the shape of the lane.

To summarise, a test T contains the task of driving along a path P through a road network N, which contains roads that are composed of road segments that are made up of lanes. What follows are formalisations of each of these components.

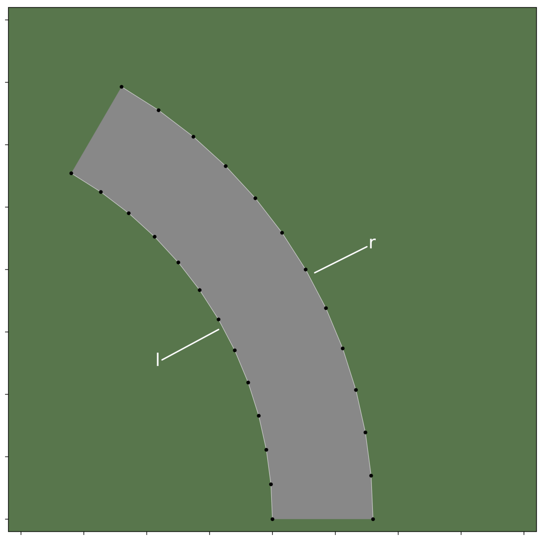


Figure 5: A simple turning lane with the vertices and edges highlighted.

4.2.1 Lanes

The most basic building block of road networks are lanes. A lane is defined as a pair of polylines. This is similar to how CommonRoad [101] defines "lanelets."

Definition 4.1 (D4.1)**.** A lane $L = (l, r)$ is a pair of polylines $l = (lv_1, ..., lv_m)$ and $r = (rv_1, ..., rv_m)$.

One such lane with its polyline vertices highlighted can be seen in Figure 5. Note that the polyline pair is required to have the same amount of vertices. Furthermore, it is required that the polylines do not intersect and that the distance between each vertex pair (lv_i, rv_i) is equal to any other pair (lv_j, rv_j). This ensures lanes remain the same width.

4.2.2 Road Segments

Road segments are sequences of left and right lanes. They are comparable to parts of roads in real life like a turn or a stretch of straight road. These segments are combined to create a road. Each segment contains a number of lanes on the left side of the road, and a number of lanes on the right side. More formally, a road segment is defined as:

Definition 4.2 (D4.2)**.** A road segment is a collection of lanes $S = (L_1, ..., L_n)$ with a dividing index d_S. Any $L_i, i < d_S$ is a lane with direction $left$ and any $L_i, i \geq d_S$ is a lane with direction $right$.

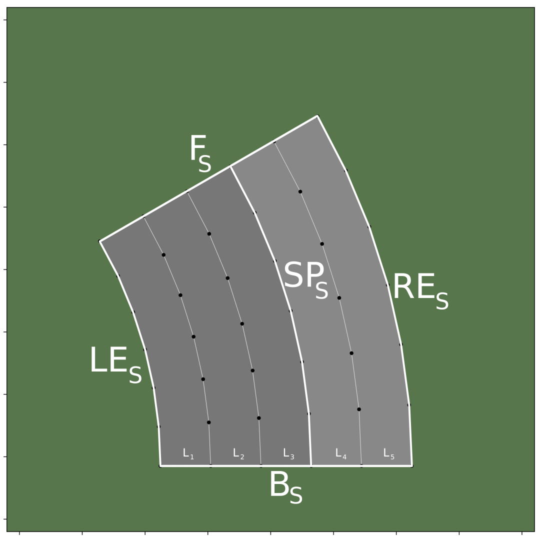


Figure 6: An example road segment with five lanes, split into left and right at the fourth lane.

One example segment with five lanes total, dividing index of four, can be seen in Figure 6. My thesis focuses on roads with one lane on each side, but the example shows how the model can be used beyond that. These segments are defined "right forward", meaning the right sequence of lanes is driven on in direction of the polyline vertices, whereas the lanes on the left are driven on in the opposite order. An illustration of this can be seen in Figure 7. Besides the mere sequence of lanes with a divider, there are a few properties of road segments derived from their components. For one, a road segment's "spine" represents the polyline dividing left lanes from right lanes. It is used to verify intersections later, as road segments that overlap only in parts and do not cleanly intersect at the spines are considered invalid. See Section 4.10. It can be seen labelled as SP_S in Figure 6.

Definition 4.3 (D4.3)**.** For a road segment $S = (L_1, ..., L_n)$ with dividing index d_S, the line SP_S is the segment's spine. It is the polyline dividing left lanes from right lanes. For segments with only left lanes, $d_S = n + 1$, it is the right edge r_n of the lane $L_n = (l_n, r_n)$. For other segments, it is the left edge l_{d_S} of the lane $L_{d_S} = (l_{d_S}, r_{d_S})$.

A road segment's shape is entirely defined by combining the shape of the individual lanes that make up the segment. The components of this shape are defined as follows:

Definition 4.4 (D4.4)**.** The front line F_S of a segment $S = (L_1, ..., L_n)$, where each lane is a pair of polylines $L_i = (l_i, r_i)$ with vertex count m, is the tip of each lane edge combined into one line. It's labelled in Figure 6 as F_S. Formally, $F_S = (lv_{1,m}, ..., lv_{n,m}, rv_{n,m})$ where $lv_{i,j}, rv_{i,j}$ are the j-th vertex of the i-th lane's left and right edge respectively.

Definition 4.5 (D4.5)**.** The back line B_S of a segment $S = (L_1, ..., L_n)$, where each lane is a pair of polylines $L_i = (l_i, r_i)$ with vertex count m, is the start of each lane edge combined into one line. It's labelled in Figure 6 as B_S. Formally, $B_S = (lv_{1,1}, ..., lv_{n,1}, rv_{n,1})$ where $lv_{i,j}, rv_{i,j}$ are the j-th vertex of the i-th lane's left and right edge respectively.

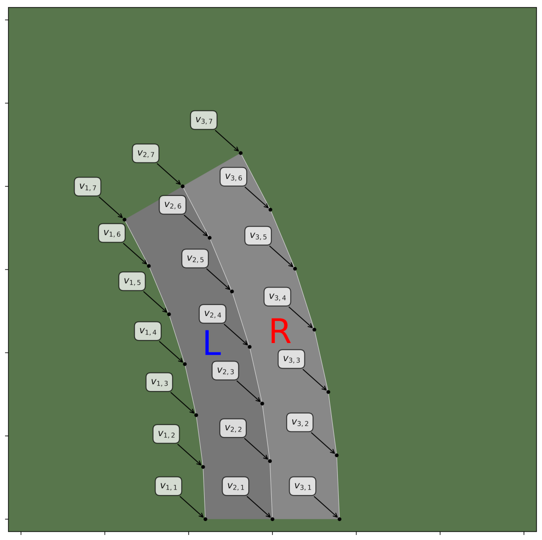


Figure 7: Example road segment highlighting lane direction's relation to vertex order.

Additionally, segments have leftmost and rightmost edges, seen as LE_S and RE_S in Figure 6.

Definition 4.6 (D4.6)**.** A road segment $S = (L_1, ..., L_n)$ has the left edge $LE_S = l_1$, where l_1 is the left edge of the lane $L_1 = (l_1, r_1)$.

Definition 4.7 (D4.7)**.** A road segment $S = (L_1, ..., L_n)$ has the right edge $RE_S = r_n$, where r_n is the right edge of the lane $L_n = (l_n, r_n)$.

Definition 4.8 (D4.8)**.** The polygon PL_S of a road segment S is constructed by joining the left edge LE_S, the back line B_S, the right edge RE_S, and the front line F_S into one polygon PL_S.

To fulfill the requirement of road networks having gapless, continuous roads, these road segments have to meet several criteria for them to be easily combine-able into valid networks. They ensure lanes within segments align and that segments align with other segments. More specifically, the requirements are:

lane-align: Any adjacent pair of lanes is required to share a polyline. Formally, for any $L_i = (l_i, r_i), L_j = (l_j, r_j), i = j + 1$ of a segment $S = (L_1, ..., L_n)$, it has to hold that $r_i = l_j$.

front-straight: The front of each lane must form a straight line. That is, for a segment $S = (L_1 = (l_1, r_1), ..., L_n = (l_n, r_n))$ it is required that the polyline F_S is perfectly straight.

back-straight: Similarly, the back of each lane must form a straight line. That is, the polyline B_S must also be perfectly straight.

Road segments meeting these requirements can then be combined to form roads.

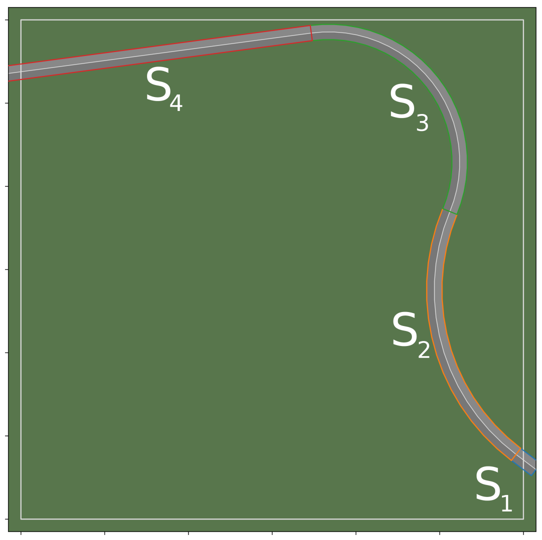


Figure 8: An example road showing how multiple road segments get combined into one continuous road.

4.2.3 Roads

Further building up my testing components, roads are defined as a linear sequence of road segments $R = (S_1, ..., S_n)$. Taking a road from the real world, for example, it can be understood to be a straight segment of a certain length, followed by a turn of a certain angle, followed by another turn, and so on. Each of these individually identifiable parts would be one road segment, and the entire road the combination of them. Since my networks are required to be gapless, this combination of segments is subject to a few restrictions, however:

back-front-align: Any adjacent pair of segments $S_i, S_j, i = j + 1$ in a road $R = (S_1, ..., S_n)$ must align at their respective back and front lines. Formally, $F_{S_i} = B_{S_j}$.

This requirement ensures any lane in a road segment can be driven on without interruption and any lane can be switched to/from without interruption. An example road with the individual segments that it is composed of highlighted can be seen in Figure 8.

Additionally, to prevent roads from intersecting with themselves, it is required that no two segments within the same road intersect each other:

non-intersect: For segments in a road $R = (S_1, ..., S_n)$ with respective polygons $(PL_{S_1}, ..., PL_{S_n})$ it must hold that every pair $(PL_{S_i}, PL_{S_j}), i \neq j$ do not intersect.

Given the nature of roads, it's easy to see how properties similar to those of road segments can be derived: A road $R = (S_1, ..., S_n)$ has a spine SP_R that is created by concatenating each segment's spine SP_{S_i}, the front line F_R is the front line of the frontmost segment F_{S_n}, and so on.

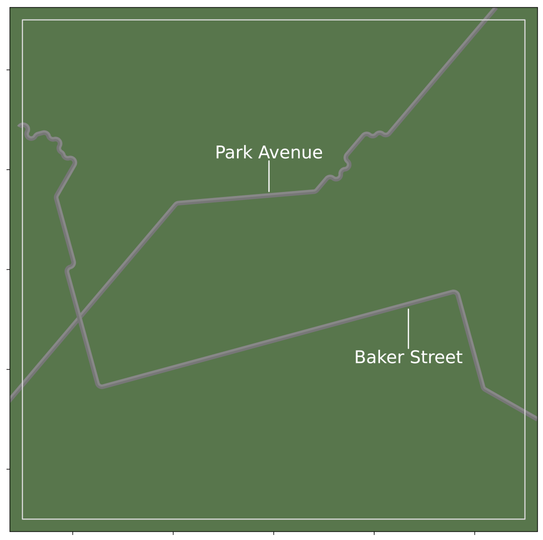


Figure 9: An example road network showing how networks are formed analogous to how streets in real life intersect.

4.2.4 Road Networks

Road networks can be formed by combining multiple roads $N = (R_1, ..., R_n)$. Analogous to real life, a network of roads can usually be decomposed into individual roads. A city might have a "Baker Street" and a "Park Avenue" that intersect to form a network. See Figure 9 for an — albeit contrived — illustration of this that shows two such roads intersecting.

So far, reachability has not been a relevant concept; within a single road, because they are assumed to be gapless, every segment is reachable from the other as one merely needs to drive up/down the lane until they reach it. With multiple roads, this is not the case. When looking for a path through a network, reachability information is needed, however, to ensure the path obtained is one the ego-vehicle can actually drive on without having to leave the network. For a road network, I define a reachability graph that contains reachability information for each segment of every road in N. Segments that share an edge in this graph are reachable without having to leave the road network (i.e. not having to drive outside the space induced by the road segments.) Formally:

Definition 4.9 (D4.9)**.** For a road network $N = (R_1, ..., R_n)$ the reachability graph $G_N = (V_N, E_N)$ is a directed and possibly cyclical graph. The nodes are composed of each road's segments:

$$V_N = \bigcup_{i=1}^{n} R_i$$

The list of edges is constructed by inserting one for each adjacent pair of segments in a single road and each pair of segments from different roads that intersect, so:

$$E_N = \{(S_{i,j}, S_{i,k}), |j - k| = 1\} \cup \{(S_{p,j}, S_{q,k}), p \neq q \wedge intersecting(S_{p,j}, S_{q,k})\}$$

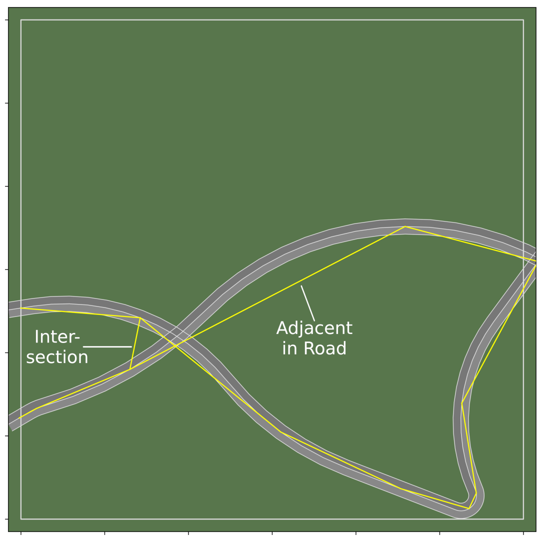


Figure 10: An example road network with its reachability graph superimposed. A yellow line between two segments means they share an edge in the reachability graph. Most edges are from segments being adjacent in the road, but one is due to two segments intersecting.

Where $S_{i,j}$ refers to the j-th segment in the i-th road of N and *intersecting* is a predicate that is true for two segments that are intersecting — its definition can be found in Definition D4.10.

Figure 10 shows an example network of two roads with their reachability edges laid over each segment. Essentially, any segment adjacent in a road, or intersecting between roads are considered reachable. Finding a path from one segment to another, regardless of which road it is in, is then equivalent to finding a path through the reachability graph. Note that this graph can easily become cyclical as soon as one intersecting pair is reachable from another intersecting pair. Finding a path from one segment to another, such as it is done when a path from start to goal of a test is searched, can be achieved using any of the established search algorithms for graphs [102].

What it means for two segments to be intersecting is intuitively clear: One segment overlaps the other such that they form an intersection. However, segments merely overlapping is an insufficient criterion, as segments could also partially overlap without forming a full crossing. See Figure 11a for an example of this: Two segments overlap but only partially, causing a shape of road that was considered invalid in this approach. To formalise this notion:

Definition 4.10 (D4.10)**.** The predicate $intersecting(S_i, S_j)$ for two road segments $S_i \in R_p$, $S_j \in R_q$ is true iff all of the following criteria are met:

- They are not members of the same road, i.e. $R_p \neq R_q$.
- Their spines SP_{S_i} and SP_{S_j} intersect at one point.

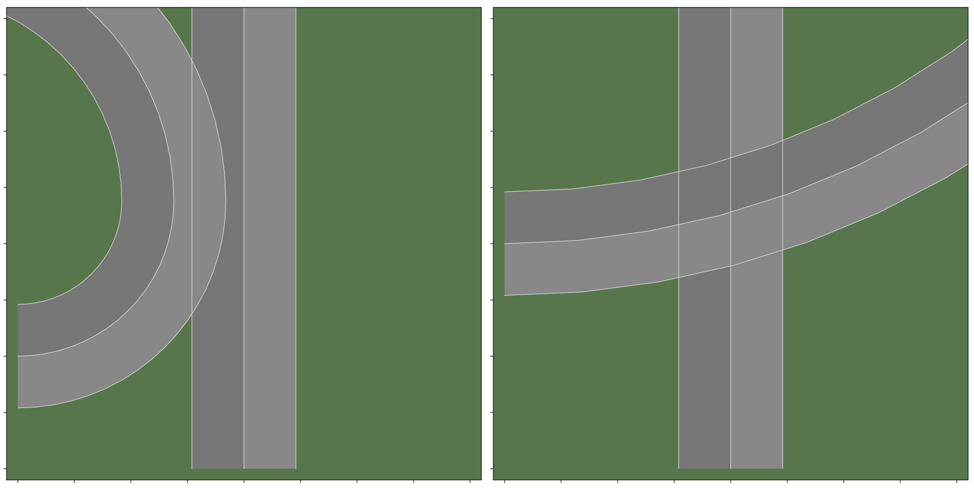

(a) Invalid partial overlap (b) Valid clean intersection

Figure 11: This figure shows an example of an invalid and valid intersection. The left one is considered invalid because the lane only partially overlaps a lane in another road, whereas the ones on the right cleanly intersect. Notice the difference in spine intersections: On the left, the segment polygons overlap, but the spines don't intersect. On the right, the segment polygons overlap and the spines intersect at exactly one point.

- Their front lines F_{S_i} and F_{S_j} must not intersect any other segment.
- Their back lines B_{S_i} and B_{S_j} must not intersect any other segment.

This definition encapsulates the notion that two segments must "cleanly" intersect. An illustration of such a clean intersection with the components used in that decision can be seen in Figure 11b. Besides this requirement on intersections, it is also required that every road in a network is reachable from another. The path of a test is required to contain only reachable transitions. It would never include roads that are not reachable from the chosen starting point, making them effectively irrelevant. To prevent this, road networks must maintain that every road is reachable from every road:

roads-reachable: For a network $N = (R_1 = (S_{1,1}, ..., S_{1,p}), ..., R_n = (S_{n,1}, ..., S_{n,q}))$ it has to hold that for every pair of distinct roads road $R_i = (S_{i,1}, ..., S_{i,p}), R_j = (S_{j,1}, ..., S_{j,q})$ there exists a path between every $S_{i,k} \in R_i$ and every $S_{j,l} \in R_j$.

4.2.5 Boundary

While the road networks, roads, and segments have to abide by certain restrictions, they could theoretically be infinite in size. This is in theory not a problem, but in practice, executing a test that could be infinitely big obvious drawbacks such as taking infinitely long to execute. How tests are generated is also complicated by the lack of a clear end. Additionally, the amount of space one has within a simulator is usually bounded by technical limitations. This motivates restricting tests to some finite space that limits amount of road segments that can possibly fit into that space and the amount of time required to drive on them. A boundary could take any shape, but for simplicity, it was restricted to being a square in this approach. Formally:

Definition 4.11 (D4.11)**.** A test boundary B is a finite polygon with the shape of a square.

Definition 4.12 (D4.12)**.** For a road segment S and boundary B, the predicate $inbounds(S, B)$ is true iff the segment polygon PL_S intersects B.

Definition 4.13 (D4.13)**.** For a road segment S and boundary B, the predicate $edgeseg(S, B)$ is true iff the segment polygon PL_S intersects the edge polyline of B.

An example road network with the boundary highlighted can be seen in Figure 4. With this boundary, some additional properties must hold for road networks. Most obviously, the road segments need to actually be within the boundary imposed on the network. A less obvious requirement that follows for in my approach is that roads need to start and end at the boundary edge. The reasoning behind this is mainly to specify clear criteria of where generation has to start and end.

segs-inbounds: For a road network $N = (R_1, ..., R_n)$, the set S_N of all segments in every road $R_i \in N$, and the boundary B it must hold that $\forall S_i\ inS_N : inbounds(S, B)$.

roads-edge: For a road network $N = (R_1, ..., R_n)$ and boundary B, it has to hold that $\forall R_i = (S_1, ..., S_p) \in N, edgeseg(S_1, B) \wedge edgeseg(S_p, B)$.

4.2.6 Tests

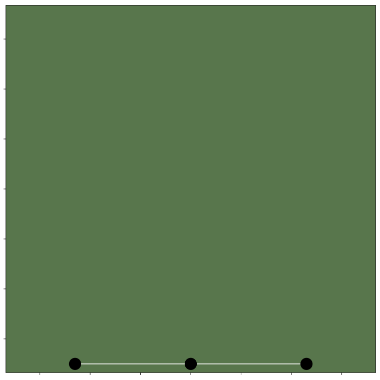
As outlined in the beginning of this section, a test is essentially the task of driving through a network along a certain path. With all these components formalised, the definition of a test also becomes more clear:

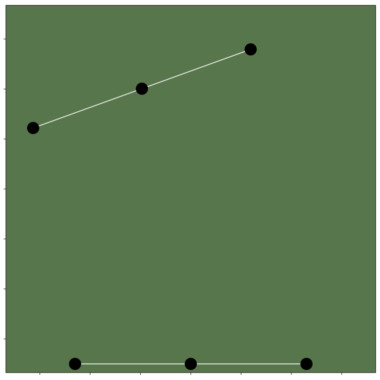
Definition 4.14 (D4.14)**.** A test $T = (N, B, P)$ is a tuple combining a road network N, a boundary polygon B, and a target path $P \subset S_N$, where S_N is the set of road segments contained in the roads of N.

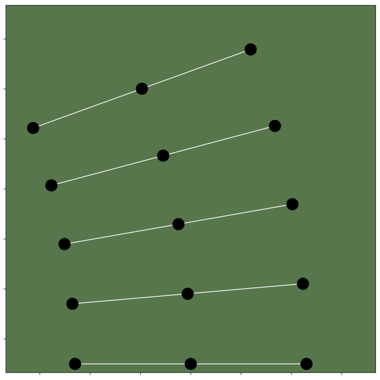
The path P is the sequence of road segments to be traversed from the start to the goal of the test. It is required that each adjacent pair in this sequence is actually reachable:

path-reachable: For a path $P = (S_1, ..., S_p)$ through network N, it is required that $\forall S_i, S_j \in P, |i - j| = 1 : (S_i, S_j) \in E_N$, where E_N is the set of edges in N's reachability graph.

This concludes definition of the test representation.


(a) Generate back line


(b) Transform back to create front line


(c) Interpolate intermediate lines

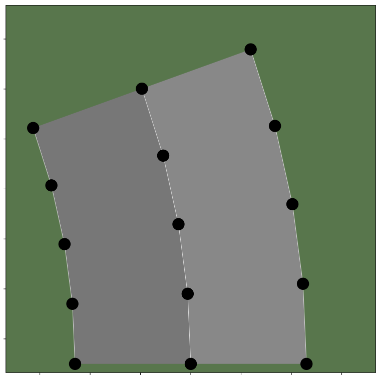

(d) Construct lanes from vertices

Figure 12: Here, the method with which road segments are generated is shown. Segments are generated by first constructing the back line they're supposed to have, applying a transformation to that line to create the front line, interpolating the transition between those, and using the result to finally construct the segment's lanes.

4.3 Generation & Modification

It is now possible to describe how such tests are exactly generated. This follows the same pattern as in the definitions above, "bottom-up": Road segments are generated, they get combined to form roads, which are combined to form networks, through which a path is found — all within a fixed boundary.

4.3.1 Road Segment Generation

To make road segments that are valid under the constraints described above, the generation algorithm starts by generating the back line B_S of the to-be-generated segment S. This line is constructed from the desired lane count, the dividing index d_S, and the supposed width lanes. The algorithm then places a vertex for each left side of a lane, and a final one for the right one of the right most lane along a straight polyline, shown in Figure 12a. This line is then transformed according to an affine transformation matrix to create the front line F_S. Figure 12b shows how B_S is translated and rotated to create the front line for a left turn. Then, to smooth the transition between front and back, the algorithm interpolates intermediate steps between back and front line. The resulting lines can be seen in in Figure 12c. Finally, the vertices of each line are used to construct polylines of the left and right edges of the desired lanes. This and the resulting road segment are shown in Figure 12d. It is easy to see how this generation method maintains the requirements for road segments:

lane-align: Since the edges of adjacent lanes $L_i, L_j, |i - j| = 1$ are constructed from the same vertices in the generated lines from B_S to F_S, they naturally share an edge.

back-straight: B_S is generated to be straight.

front-straight: F_S is generated to be straight.

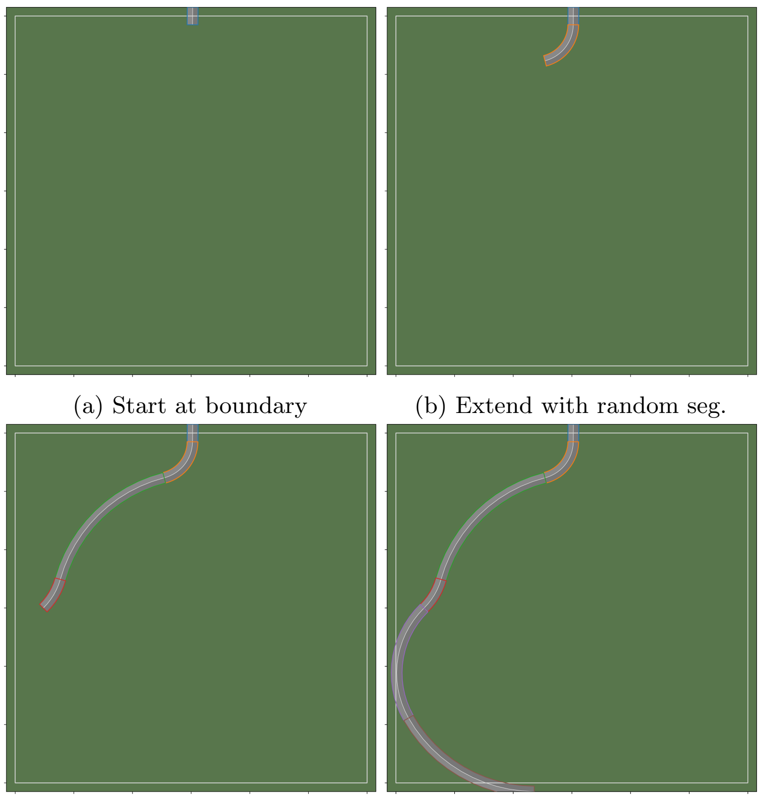
This generation algorithm is parameterised through the lane width w that defines the distance between vertices along the back line and the affine transformation matrix applied to generate the front line. Within the scope of this thesis, w was fixed to prevent lanes of variable width, as they would complicate lane keeping evaluation. Evaluating the impact of varying lane width is left for future work. My generation picked transformation matrices from the following types:

1. Straight transformations, which merely move the back line forwards a certain amount of metres to form straight stretches of road.
2. Left turns, which rotate the back line to the left by $n < 0$ degrees around a pivot left of the back line pv.
3. Right turns, which rotate the back line to the right by $n > 0$ degrees around a pivot right of the back line pv.

More complicated shapes can be created by combining multiple such segments. For example, an "S"-shaped windy road can be made by combining a left and a right turn.

4.3.2 Road Generation

With the ability to generate valid road segments, and roads being a linear sequence of road segments, road generation can be achieved by appending randomly generated road segments. Since road networks are restricted to a boundary polygon B, this method is started at a random point B_{START} along the boundary edge of B and placing a straight segment there. This segment is rotated towards the centroid of the boundary polygon to ensure generation is headed into, not out of, B. Figure 13a shows the initial state of generation, having placed a straight segment at a random point along B's edge. From that point on, the road is built up by appending segments to the current end of the road, using the front line of the ending segment as the back line during generation of the next segment. This is repeated until a segment is generated that is no longer in bounds. Some intermediate states of this generation are illustrated in Figure 13. Contrary to segment generation, with this algorithm, it's less clear how the generated roads follow the requirements expected of roads. For some, it is fairly obvious:


(a) Start at boundary (b) Extend with random seg.

(c) Extend with random seg. (d) Finish when out of boundary

Figure 13: These images show sample states during road generation. Roads are generated one segment after another, but the series of images leaves out some intermediate steps for brevity. Generation is started from a random point at the boundary. From that point, the road gets randomly extended until the boundary is exited. If generation creates an invalid road, the algorithm backtracks.

back-front-align: Front and back of each segment will align because a segment is generated with the back being the front of the previous segment.

segs-inbouds: Generation starts at the edge of the boundary B pointing inwards and finishes once a segment would be out of bounds. Therefore, for all generated segments $S_i, inbounds(S, B)$ is true.

roads-edge: Similarly, since generation starts and ends at the boundary, this property is fulfilled.

The harder-to-meet requirement is the **non-intersect** property, which requires roads to not intersect with themselves. This was first achieved by testing a road for self-intersections after generation. If a road was found to be self-intersecting, it was discarded and generation was attempted again. However, during preliminary study, this led to *very* long generation times as the algorithm easily expanded a road back in on itself or into a corner. This was improved upon by checking for self-intersections after each segment is added, and backtracking if the recent addition caused an intersection. If no segment to append was found that does not cause self-intersections, the algorithm backtracked further. This still left the algorithm susceptible to generate roads into a corner that took a long time to backtrack out of. To make this less likely, the algorithm was modified to always move towards a point B_{GOAL} that is defined as the diametric opposite of B_{START}. Any addition to the road whose front line increased the distance to B_{GOAL} was discarded, lowering the likelihood of the algorithm backing up on itself.

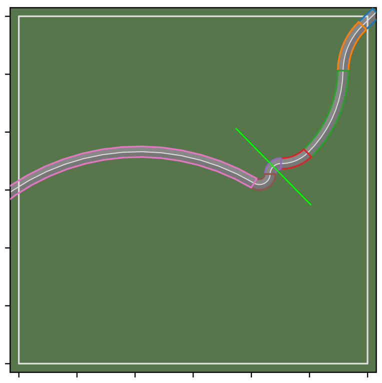
4.3.3 Test Generation

To generate a test from a network N with boundary B, one merely needs to find a path P through to complete the tuple $T = (N, B, P)$. Because of the requirements a road network must meet to be valid, there exists a path between every road segment in N. Therefore, any pair of segments would work to select a path. In my approach, however, a path is chosen by randomly sampling distinct pairs (S_S, S_G) of segments that lie on the edge of B, then sampling a random set of paths from S_S to S_G, and picking the longest between them. This was done to maximise how much of the network is actually traversed during testing. Note that, since the reachability graph is cyclical, the amount of possible paths from S_S to S_G would be infinite. Instead, only simple paths are considered, which include a segment at most once. Even so, with sufficiently complicated networks, there's a sort of path explosion that is mitigated by only evaluating a randomised sample — the amount of simple paths for a graph of order n is $O(n!)$ [102].

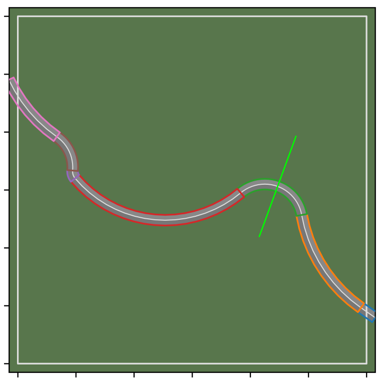
Once a path is found, the test can be created. This is how the test shown earlier in Figure 4 was generated.

4.4 Genetic Algorithm

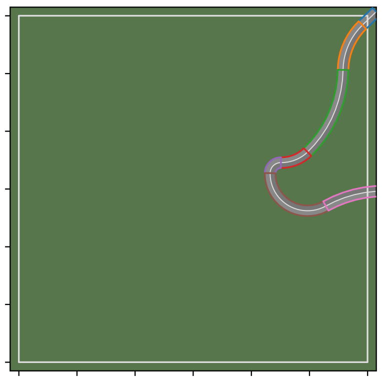
As outlined before, my genetic algorithm evolves a population of tests towards a suite to get the vehicle to break out of the lane bounds. In the terms defined above, the GA maintains a population of tests $I = (T_1, ..., T_n)$ that is initialised randomly. In each generation, every test $T_i \in I$ is executed in a driving simulator with an AI as the test subject. The driving behaviour observed is used to rank the fitness of each test in the population. For fitness, the GA uses the maximum distance to the lane centre observed during testing. This means a test's fitness is based on how far it got the vehicle away from the lane centre. A more formal definition of this is given in Section 4.6.2. The GA then follows standard GA behaviour, randomly selecting mates with preference to fit ones and producing offspring of them for the next generation. The best test from the current generation is always copied into the next one, maintaining an elitism of one. How tests are crossed over and mutated is elaborated, as these operations are designed by me and are specific to this thesis.



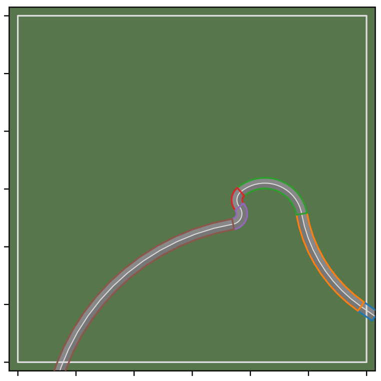
(a) Pick random joint in left parent



(b) Pick random joint in right parent



(c) Combine first offspring



(d) Combine second offspring

Figure 14: These images show how the *join*() operator works. Given two parent roads, a random joint segment is chosen for each road. Parent roads are then split into sub-roads before and after the joint and offspring is combined from the resulting four sub-roads.

4.4.1 Crossover (Join)

For crossing over tests, I define two operators. They work on different levels of the test, one on roads and one on the entire road network. To cross over roads, I define a so-called "join" operator: it crosses over two roads $R_{1,i}$, $R_{2,j}$ from two networks $N_1 = (R_{1,1}, ..., R_{1,p}), N_2 = (R_{2,1}, ..., R_{2,q})$ by picking two random joint segments $S_1 \in R_{1,i}, S_2 \in R_{2,j}$, splitting $R_{1,i}$ and $R_{2,j}$ at the respective joints. This results in "sub-roads" of each road that contain the segments before and after the joint. It then combines these four sub-roads such that no two come from the same road. An illustration of this operation can be seen in Figure 14. More formally, the join operator is defined as:

Definition 4.15 (D4.15)**.** The $join(A, B)$ operator for two roads $A = (S_{A,1}, ..., S_{A,p})$ and $B = (S_{B,1}, ..., S_{B,q})$ produces two offspring roads A' and B' with the following algorithm:

1. Pick random $S_{A,i} \in A, i < p$

2. Pick random $S_{B,j} \in B, j < q$
3. Split A into $A_{pre} = (S_{A,1}, ..., S_{A,i})$, $A_{post} = (S_{A,i+1}, ..., S_{A,p})$
4. Split B into $B_{pre} = (S_{B,1}, ..., S_{B,j})$, $B_{post} = (S_{B,j+1}, ..., S_{B,q})$
5. Produce $A' = A_{pre} \cup B_{post}$
6. Produce $B' = B_{pre} \cup A_{post}$

To apply this operator to networks, not just single roads, random roads are picked from the networks N_A and N_B to join, with the unused roads carried over into A' and B' untouched.

A very important aspect of this, whose precise definition is omitted for brevity, is that the segment geometry needs to be re-translated for both A_{post} and B_{post} as they get attached to B_{pre} and A_{pre} respectively. This is where my approach reaps the benefits of polylines. As described in the sections defining roads and road networks (Section 4.2.3 and Section 4.2.4), certain requirements must be met to be considered a valid road network. The join operator, however, can easily produce invalid offspring: The resulting roads could self-intersect, partially overlap with another road in the network, not reach the boundary, and so on. If this happens, the operation is considered failed and the offspring will not be added to the next generation. Upon failure, operations are retried with a probability of giving up that increases per failure.

4.4.2 Crossover (Merge)

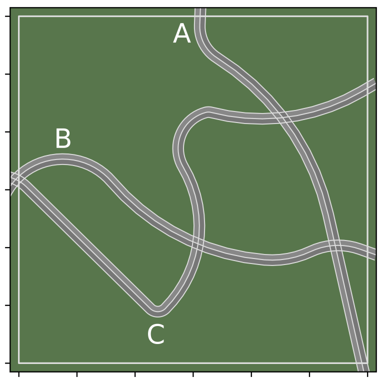
The second operator for crossovers works on road networks as a whole. It takes a random subset of roads from each parent network, produces two new networks by merging the selected roads into one and the leftover roads into another network. An example of the merge operation is illustrated in Figure 15. More specifically:

Definition 4.16 (D4.16)**.** The $merge(A, B)$ operator for two networks $A = (R_{A,1}, ..., R_{A,p}), B = (R_{B,1}, ..., R_{B,q})$ produces two offspring networks A' and B' using the following algorithm:

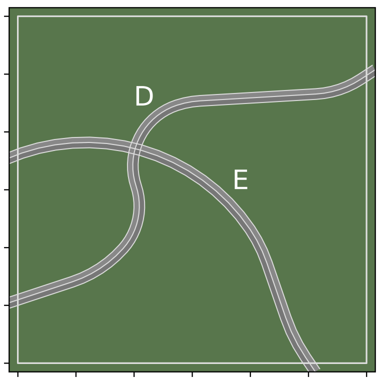
1. Sample randomised subset $A_{pick} \subseteq A$
2. Define $A_{left} = A \setminus A_{pick}$
3. Sample randomised subset $B_{pick} \subseteq B$
4. Define $B_{left} = B \setminus B_{pick}$
5. Produce $A' = A_{pick} \cup B_{pick}$
6. Produce $B' = A_{left} \cup B_{left}$

Note that, if $A_{pick} = A$ and/or $B_{pick} = B$, B' can end up being merely A_{left}, B_{left} or even completely empty. This can easily happen if A and/or B contain only one road, for example. Empty networks are considered invalid.

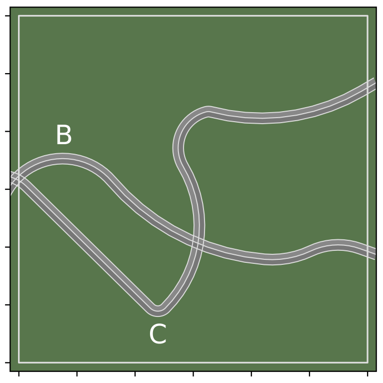
Like $join()$, this operator can end up producing invalid results. Networks in which not every road is reachable from another, partial overlaps, and so on. The same applies here: Invalid offspring is not carried over into the next generation and the GA re-attempts application of the operator with a probability to give up that increases per failure.



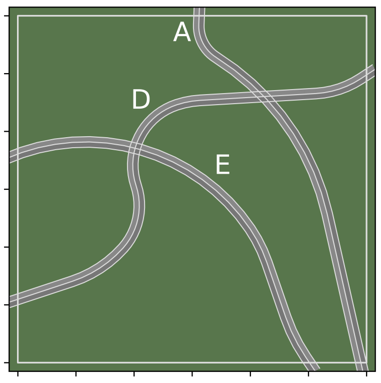
(a) Pick random roads from left parent, A here



(b) Pick random roads from right parent, D & E here



(c) Combine first offspring



(d) Combine second offspring

Figure 15: This series of images shows the $merge()$ operator being applied. It takes random subsets of the parent road networks and merges the chosen and leftover roads into two offspring networks.

4.4.3 Mutation (Segment replacement)

Mutation is done by picking a random segment S, creating a random replacement segment S', and replacing it in the target network. See Figure 16 for an example.

Definition 4.17 (D4.17)**.** The $mutate(N)$ operator for a network $N = (R_1, ..., R_n)$ works as follows:

1. Pick random $S_{i,j} \in R_i, R_i \in N$
2. Generate random replacement $S'_{i,j}$ segment
3. Replace $S_{i,j} \in R_i$ with $S'_{i,j}$ to create R'_i
4. Replace $R_i \in N$ with R'_i to create N'

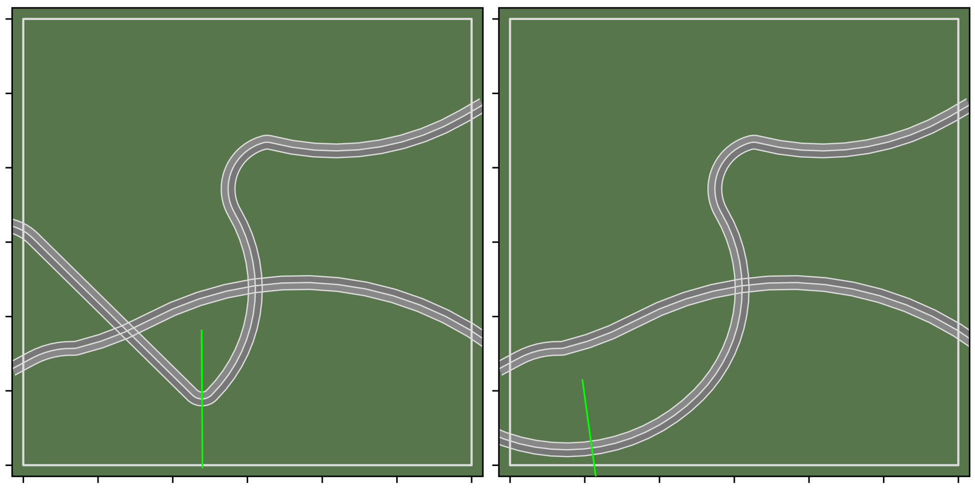

(a) Pick random segment (b) Replace with another random segment

Figure 16: This figure shows an example application of the mutation operator replacing a random segment in a road network with another random segment.

Similar to $join()$, this operation needs to translate geometry after $S_{i,j}$ to fit onto $S'_{i,j}$. If N' is not valid according to my requirements, it will not be carried over into the next generation.

4.4.4 Notes on the Genetic Algorithm

The definitions above all have the potential to produce invalid offspring. However, they are also randomised. Therefore, it is possible that an operator fails one time, but, applied again to the same pair, succeeds. To deal with this, the GA tries an operator multiple times, with a random chance to give up on it that increases with each failure. An alternative to this would be exhausting every possible option in these crossovers to find a successful pair, or rule out the existence of one, but this was not used to save computation time. The $join()$ operator in particular would require enumerating the Cartesian product between two roads A and B. It would produce $O(n^2)$ new networks whilst searching for a valid one.

With regards to having multiple options for crossing over two networks: The GA is configured with a probability of which one to pick. For example, it can be set to choose $join()$ 30% of the time and, in turn, $merge()$ 70% of the time. If the chosen operation fails, the other one is not attempted.

Once operations do succeed, the GA produces tests from the crossed over/mutated networks by simply generating one for each network as described in Section 4.3.3. This is done until enough offspring has been collected for the next generation. If, due to failures, too few offspring were created, the next generation is padded with the most fit individuals of the last one to make up for the lacking offspring.

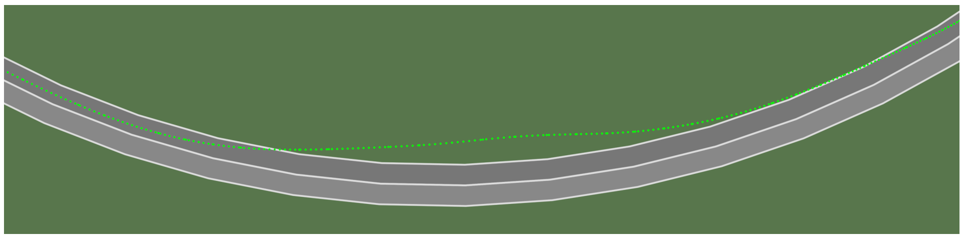


Figure 17: An example vehicle trace. The green dots show each sample of the vehicle's position laid over the road it's driving on. The vehicle breaking lane is clearly visible.

4.5 Test Execution

My tests are imported into a driving simulator and are then run by having an AI perform the driving task defined as part of the test within the simulation. The simulator for this has to accept roads in a format similar to polylines to be compatible with my approach. As output, it needs to be able to provide the vehicle's position in the test at a rate that makes analysis of that meaningful. In other words, if the simulator can only provide the vehicle's position every few seconds, test execution might miss times the vehicle went outside the lane for less than that time.

4.6 Metrics

Tests in my context don't produce binary pass/fail results. Instead, they provide data about how many times the vehicle broke out of lane bounds, its distance to the lane centre over time, the speed it went out of bounds with, and so on. One can use this data to define binary pass/fail criteria such as never having more than a certain amount of times the vehicle broke out of bounds, but, during executions, my tests simply let the vehicle perform the driving task and gather metrics about its performance. The only hard failure is the vehicle not finishing the test in a certain time. This is to prevent a vehicle which, for whatever reason, cannot reach the goal halting test suite evaluation by taking infinite time. In early testing, the vehicle some times failed to start and held up other test executions by standing still. This is prevented by putting a hard (but generous) cap on how long a test can take. Otherwise, all data is gathered from the vehicle's trace during the test.

4.6.1 Trace Analysis

In my work I consider the vehicle trace as the sequence of its positions during execution and use it to compute metrics such the distance to the lane centre and judge whether it is within the bounds of the lane. One such trace, showing the vehicle's path along the lane and where it goes out of bounds, can be seen in Figure 17.

Definition 4.18 (D4.18)**.** A vehicle trace $v_T = (p_1, ..., p_n)$ is an ordered sequence of coordinate pairs $p_i = (x_i, y_i)$ that captures a vehicle's position over time during execution of a test T.

It is important to highlight that a vehicle is represented as a point here. This is, of course, an oversimplification, as vehicles take up space. The exact bounding box of the vehicle in the simulator was not provided, however. Test evaluation instead relied upon the position of the centre of the ego-vehicle's rear axle. While oversimplified, this only helps the test subject, because it can go partially outside the lane without the test evaluation noticing. The vehicle has to go outside the lane beyond the centre of its rear axle to count.

For each point p_i, one can easily calculate the distance to the lane centre by computing their geometric distance. To test whether a point p_i is within the lane bounds, my algorithm searches for any lane's polygon that contains p_i. More formally:

Definition 4.19 (D4.19)**.** Let p be a point and N be a network. The predicate $inlane(p, N)$ is true iff there exists an $L \in L_N$ whose lane polygon PL_L contains p. Here, L_N refers to the set of all lanes contained in all road segments of all roads in N.

This predicate is used to test if a vehicle is within lane bounds. Note that the predicate tests for the point being in the *lane* polygon, not the road. Driving onto the other lane — into oncoming traffic — is therefore also not considered to be $inlane()$. The actual out of bounds failure count of a test is not the mere amount of points for which $inline(p, N)$ is false, however. If a vehicle breaks out of lane and stays out of lane for longer than the interval between snapshots of its position, then the trace would end up with multiple points for what is intuitively the same failure. Instead, the trace is analysed for sequences of points in which the vehicle was out of bounds:

Definition 4.20 (D4.20)**.** An Out of Bounds Episode of a vehicle trace $v_T = (p_1, ..., p_n)$ in a test $T = (N, B, P)$ is a sequence of points $o = (p_i, ..., p_j), 1 \leq i < j \leq n$ where it holds that $\forall p_k \in o : \neg inlane(p_k, N)$.

When evaluating test results, the actual count of out of bounds failures is the amount of distinct Out of Bounds Episodes (OBEs) found in the respective trace.

4.6.2 Lane Distance Fitness

My tests are supposed to cause as many OBEs they can. To rank a test's ability to do that, the OBE count itself is insufficient. The predicate $inlane()$ is binary and does not express how close or how far away a test is from actually causing an OBE failure. In the early stages of evolution, where there might not even be an OBE failure found in the suite, this is of particular importance. To know which tests are likely to cause OBE failures, the GA needs a continuous metric that can be used to rank tests within the suite. To this end, I define a fitness function that measures the maximum distance from the target lane to capture how close a test is to OBE failures.

Definition 4.21 (D4.21)**.** Given a test $T = (N, B, P)$ and a vehicle trace $v = (t_1, ..., t_n)$ the *lanedist* fitness is defined as:

$$lanedist(T, v) = \max dist(v_i, P), v_i \in v$$

Where $dist(p, P)$ is the shortest distance of a point p to the centre lane in path P.

With this metric, tests are rewarded for causing the vehicle to move away from the lane centre as much as possible, even if no actual OBE failure is caused yet. The decision to use the maximum lane distance over something like the average was based on the assumption that OBEs are the exception, not the rule, and that vehicles would mostly stay in lane. Using the average lane distance in v would therefore be liable to even out short moments the vehicle was close to an OBEs.

4.6.3 Suite Diversity

An interesting aspect of the evolved test suites is their diversity. Population diversity expresses how distinct individuals are from each other. The GA might produce a test suite with n OBEs failures, but all those failures share the same cause. The suite would therefore only expose one flaw in the vehicle. Measuring the diversity of a suite can give more insights as to how unique individual failures might be. Additionally, besides failures, a diverse suite also serves to verify more behaviour of the ego-vehicle, because the vehicle is given different driving tasks. A monotonous suite would blind the tester to both possible faults, but also what the vehicle actually gets right.

In my approach, diversity is comparable with the concept of coverage; road networks are made up of roads which are in turn made up of road segments. How diverse a test/suite is, is synonymous with how well it covers possible segments. The more types of segments segments are contained within a suite, the higher the diversity, and the more shapes of roads the vehicle has to traverse.

Theoretically, the definition of segments D4.2 allows for uncountably infinite segments. This is obviously not feasible to cover. It's more reasonable to group segments that are similar in shape together and cover those instead. It is, intuitively, unlikely that a turn of 30 degrees would cause the vehicle to behave differently than a turn of 30.25 degrees, for example. The segment grouping I chose was similar to how the generator generates road segments, where the main factors are how far to move the front line in case of straight segments, and, in case of turns, by what angle to turn the front line around what pivot. Grouping divided these parameters and rounded them down to find the group a segment belonged to. For straight segments, their length was divided by ten. For turns, the angle was divided by 15 and the offset of the turn pivot by 5. Because the generator was limited to straight segments with a maximum length of 300 metres, turns with a pivot offset by 1 to 50 metres, and turn angles from -120 to 120, those were the limits chosen to make the set of possible groups actually finite.

Definition 4.22 (D4.22)**.** The group of a road segment S is:

$$group(S) = \begin{cases} (\lfloor r/15 \rfloor, \lfloor p/5 \rfloor) & \text{if } type = turn \\ (\lfloor l/10 \rfloor) & \text{if } type = straight \end{cases}$$

Where *type* stands for the type the segment was generated as, r and p are turn angle and pivot, and l is the length of a straight segment.

All possible groups would therefore be:

Definition 4.23 (D4.23)**.** The set of all possible segment groups G is:

$$G = \{(\lfloor r/15 \rfloor, \lfloor p/5 \rfloor) | \forall r \in [-120, 120], p \in [1, 50]\} \cup \{(\lfloor l/10 \rfloor) | \forall l \in [1, 300]\}$$

Covering all possible segments alone is a myopic metric of coverage/diversity. Road segments aren't driven on individually, after all. The vehicle enters a segment with the momentum it had from the previous one. It is easy to imagine how a vehicle that accelerated while driving down a long straight segment could enter a turn with too high a speed to stay in bounds. If the segment preceding the turn was shorter and didn't allow reaching high speeds, this wouldn't happen. Similarly, a vehicle might have oversteered in a left turn to the extent of not being able to turn in time to make a right turn. Examples like these motivate expanding the notion of coverage to not measure segments individually. It should at least cover all pairs of segments. This notion is complicated by intersections, however. They mean there are two ways to reach one segment from the other. A network could have a straight segment followed by another straight segment within the same road, but also a straight segment intersecting with another straight segment. All possible *pairs* of two segment groups is therefore:

Definition 4.24 (D4.24)**.** The set of possible segment pairs C is defined as:

$$C = \{(g_1 \rightarrow g_2) | \forall g_1, g_2 \in G\} \cup \{(g_1 \otimes g_2) | \forall g_1, g_2 \in G\}$$

Here, $\rightarrow$ represents a regular transition from g_1 to g_2 within the same road and $\otimes$ and intersection between them.

Extracting the set of pairs C_T in a test $T = (N, B, P)$ then requires traversing the path P combining each pair of adjacent segments in the path depending on how they're reached — either along the same road or through an intersection. Finally computing the diversity of a test and a suite is done as follows:

Definition 4.25 (D4.25)**.** The diversity of a test $T = (N, B, P)$ and the corresponding set of segment pairs C_T along the path P is equal to the coverage of all possible pairs :

$$testdiv(T) = \frac{|C_T|}{|C|}$$

Definition 4.26 (D4.26)**.** The diversity of a test suite $S = (T_1, ..., T_n)$ with corresponding test segment constellations $(C_{T_1}, ..., C_{T_n})$ is defined as:

$$suitediv(S) = \frac{|\bigcup_{T_i \in S} C_{T_i}|}{|C|}$$

In other words, a suite's diversity is the ratio between segment pairs covered in the suite and all possible segment pairs.

4.6.4 Auxiliary data

The vehicle trace v can be used to compute additional data about the vehicle's performance. This does not factor into fitness or diversity, but rather serves to supplement that data with knowledge that could aide diagnosing a fault. It is possible to compute:

- The speed at which a vehicle goes out of bounds from the distance between points in an Out of Bounds Episode.

- The segment group a vehicle goes out of bounds on by finding the segment S the point in a trace v immediately before an Out of Bounds Episode and compute its group.
- How long a vehicle takes to recover from being out of bounds (if at all.)

For example, knowing the vehicle went out of bounds only on left turns would obviously hint at there being a fault with its left turning behaviour. Such data is easily obtainable from the trace a test outputs.

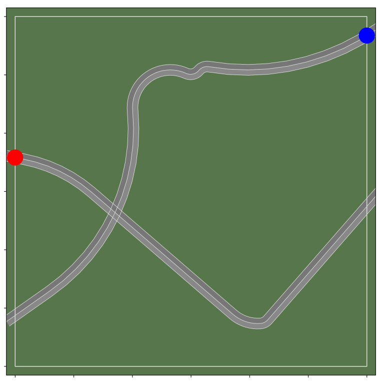
5 Implementation

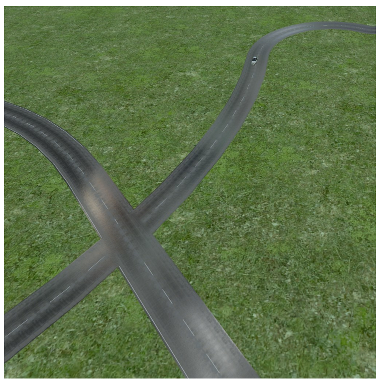
I implement my approach as a Python 3 program that offers all generation-related functions. The program contains the code implementing the model, generation method, and genetic algorithm described in the Method Section 4. Additionally, it can persist tests to disk in JSON format that encodes road networks as polylines, similar to the formalisations presented earlier. Stored tests can then be loaded and executed at will. This allows the user to generate a test suite as described earlier, save it, and re-run for regression testing and benchmarking. Additionally, functions to export tests to a simulator are implemented for my simulator of choice: BeamNG.research [67].

5.1 BeamNG.research Overview

BeamNG.drive is a driving game featuring realistic soft-body physics [92] for the vehicles the players interact with. This product has since been forked into a research-focused, freely available, version of the game that adds functionality tailored for research. Most importantly, however, it retains the realistic simulation of driving physics. This gave BeamNG.research an advantage over other simulators for autonomous vehicle testing such as CARLA [50], AirSim [89], or DeepDrive.io [90], which focus less on driving physics. These physics were given a high priority in simulator selection as my tests evaluate lane keeping. A simulator that does not accurately reproduce the factors like inertia acting upon a vehicle, the friction to the ground, acceleration and braking behaviour, and so on, would exclude many aspects relevant to lane keeping. BeamNG.research including these in its simulation made it the most appropriate simulator to my knowledge.

In addition to its physics. BeamNG.research is also highly configurable through Lua scripts. These allow definition of the environment, layout of roads in particular, and also allow the user to track the vehicle's position in the environment in real time. Executing tests in BeamNG.research was helped by these features. While the format required for roads isn't exactly like polylines, it's a format one can easily convert to and from polylines. Instead of being defined through edge polylines, one specifies a road in BeamNG.research by giving the *centre* polyline and the width of the road for each vertex therein. Converting two polylines to this format only requires forming the centre line between them by taking the centre between each vertex pair and using the distance between these vertices as the width. This was done for every road in a network to recreate a road network within BeamNG.research. An example can be seen in Figure 18 which shows a test in my representation and the same test exported to BeamNG.research. During execution, the simulator was configured to run a Lua script that retrieves the ego-vehicle's position every 250ms.


(a) Sample test


(b) Exported to BeamNG.research

(c) View from driver's seat

(d) Orbit view

Figure 18: These images show a test plotted with my program on the top left and the same test exported to BeamNG.research on the top right from a bird's eye view. Note that the image on the top right is focused on the area near the intersection of the test, so the rest of the test is not visible.

This simulator can also be set to run faster than real time, while retaining accurate physics. This helps accelerate testing, because, to drive down a 30km road at 30km/h, it doesn't actually require one hour in real time. Rather, if the hardware allows for it, the simulation can be run at twice the speed or more to require less time. If one assigns the simulator process its own dedicated working directory, executions are also entirely independent and can be run in parallel.

5.2 Test Execution

Since BeamNG.research runs as its own process and is not available as a Python library, test execution relied on interprocess communication. For every test execution, my program exports the test into BeamNG.research's format and saves them in the simulator's working directory. It then spawns a new process of the simulator and opens a communication socket. The BeamNG.research process is given the test file to load and a Lua script to run. This script is part of my implementation. It connects to the socket of my Python program and allows for communication between the two. This socket is how my program controls graceful termination of the simulator after tests and through which it receives the vehicle trace for analysis.

6 Evaluation

The evaluation of this thesis was carried out in a series of experiments. These gathered data to examine the proficiency of my algorithm with regards to generating suites that produce out of bounds episodes in various settings. Experiments were split into multiple conditions that revealed more insight into which components of my methodology contribute to the goal of generating a good test suite for out of bounds failures. Over the course of the experiments, the generation method was gradually expanded to incorporate more of the components and algorithms discussed in the methodology. In short, the experiments were to examine:

- How my generation compares to a naïve method like random generation.
- How diverse generated suites are.
- How much time effective suite generation takes.
- How varying the size of the networks affects results.
- How well the generation works for different driving styles.
- How different mutation rates affect OBEs/diversity.

Overall, the experiments followed the pattern of evolving a suite of 25 tests over 50 generations, with aspects of the generation being varied for each experimental setting. Tests were always executed in my simulator of choice: BeamNG.research [67]. Unless stated otherwise, the experiments had the GA configured to use an Elitism of at least 1 (see Section 4.1 on why this can be higher) and a mutation rate of 50%. Boundary size was $2km \times 2km$. During execution, the ego-vehicle was given a generous time limit of one second per metre of the goal path. In other terms, it had to drive a minimum speed of 3.6km/h. This was due to occasional failures in preliminary testing that caused the subject to stand still. Cases like this are handled by the timeout condition; if the vehicle does not reach the goal point by the allotted time of one second per metre to drive, it suggests something went wrong and the execution gets aborted, recording a timeout failure.

Throughout the experiments, the dependent variables that were tracked in order to evaluate the GA regarding the topics mentioned above were:

1. Out-of-Bounds Episode count.
2. Timeout failure rate.
3. On which type of segments the vehicle breaks out of bounds.

4. At what speeds the vehicle breaks out of bounds.
5. Suite diversity.
6. Generation time.
7. Test execution time.

Significance tests on this data was performed using the Mann-Whitney U test, as recommended for randomised testing procedures by Arcuri et al. [103].

6.1 Test subject

My test subject for the evaluation was the AI BeamNG.research [67] ships with. This AI works in two steps. The first is somewhat analogous to the work by Georgiou et al. [99], where the authors calculate an ideal line along a race track the driver should follow as reference. Given a start and goal to drive from and to, BeamNG.research's AI does the same. Since there's no human operating the vehicle, however, it also formulates an optimisation problem around following this line, whose solution is the acceleration/braking/and steering to use along the track. The behaviour of this AI can be parameterised with a so-called "aggression" value. This affects how much of the road the ideal line can use (i.e. how closely it stays within the road) and the budget for acceleration/braking/steering. Essentially, a lower aggression value makes the vehicle stick to the centre of the lane with more gentle acceleration/braking/steering. This information was obtained in an informal interview with the developer, but the code for said AI is distributed with the simulator.

There's a difference between BeamNG.drive's AI and a real-world autonomous vehicle AI in one key area: It has "perfect knowledge." This means, instead of having to infer possible driving space from sensor data, BeamNG.research's AI retrieves this information from the simulation. For the evaluation, this had the downside of lowering external validity of the experiments, as it's impossible to tell to what extent the behaviour is representative of any autonomous vehicle AI used in the real world. However, conversely, internal validity increases. This is because BeamNG.research's AI works with the same representation as the algorithms: Roads encoded as polylines. This removes the effects of other components in, for example, an autonomous vehicle AI based on the common approach of neural network processing [65], where drivable space is inferred from a camera feed. In such a subject, it's not clear if the subject couldn't maintain its position within the lane because it drove poorly, or because some aspect of the simulated images we have limited control over confused it. Aspects like the width & colour of lane markers, the lighting conditions, camera perspective, and so on, all factor into visual processing, but they are not captured within my algorithm's representation. Relying on a test subject which is not affected by these helps focus the evaluation on what's actually captured in my approach: How the shape of the road affects driving behaviour. It's also noteworthy that the AI, as described above, is fairly sophisticated. Making it break lane bounds, at least for low aggression values, is not an easy task. It also operates the vehicle using the actuators of the vehicle, i.e., the AI pushes down on the gas pedal and rotates the steering wheel like a player/human would. It does not rely on other functions of the simulator to move, only what the driver of the car has available.

Attempts to get a autonomous vehicle AI from the real world running for simulations have proven to be quite difficult. These AIs are usually not written to be portable, so executing them outside their target environment is already cumbersome, let alone integrating them into any simulator. An evaluation on a real-world autonomous vehicle AI is left for future work. At the time of writing, this approach is being tested with the DeepDriving model presented by Chen et al. [48]. The results of that will be published at a later date.

6.2 Experiment 1: Comparison with Random

Work in this area has so far only looked at random generation of valid roads [59]. Given a way to generate valid roads, generating them randomly is easily achieved as one merely needs to apply the generation method randomly. In Kendall et al. [59]'s work, (valid) random generation was enough to provide roads in simulation for reinforcement learning of lane keeping. For the purpose of fault detection, its effectiveness has not been demonstrated yet. As such, the first experiment looked at comparison to random generation that produces roads similar to those generated by Kendall et al. [59].

As a baseline, I used a method slightly more sophisticated than random generation alone. Random generation was implemented by generating 50 random populations of 25 tests, keeping only the best one in terms of OBE count. In other words, each suite of 25 tests was evaluated for its total OBE count and, if it exceeded the current maximum, it replaced the suite currently known as best. This was done to give random generation a chance to grow in quality over time. The simplest implementation of random generation would generate a suite of 25 tests once and use that for reference. Early testing revealed this to be trivially easy to beat, however, so the bar was raised by giving random 50 chances to improve its population.

Random generation, as described in Section 4.3.2, only returns valid roads. This is already significantly better than pure random generation, because randomly producing points that serve as polyline vertices would result in invalid roads most of the time. Combined with the aforementioned method of only keeping the best suites random produced, makes the random method I use as a baseline to beat quite sophisticated.

6.2.1 Experiment design

The experiment was split into a total of four different conditions, the main difference between them being whether roads were randomly generated, or evolved using the genetic algorithm. As a secondary criterion, the aggressiveness of the AI was varied. The default AI aggression is 1, its maximum is 2, and the minimum is 0.7. I varied these -0.25 and $+0.25$ from the norm, resulting in a “careful” driver with aggression 0.75 and a “reckless” driver with aggression 1.25. The point of these conditions was to see how stable results are across different driving behaviours.

All conditions were restricted to single-road networks only. This was due to generation only being defined for single roads, but it also allows later comparing how multiple roads affect generation. To achieve this, the GA was configured to use $merge()$ with a probability of 0. For fitness, the GA used $lanedist()$ as defined in D4.21.

Random generation used the generation algorithm as described in Section 4.3.2. Note that this ensured random generation led to valid tests, as opposed to completely random generation which might produce invalid ones.

In summary, the independent variables for these experiments were:

1. Generation method (*random*, *evolutionary*)
2. AI aggression ($careful$, *reckless*)

This leads to the following conditions:

1. *random*, $careful$
2. *random*, *reckless*
3. *evolved*, $careful$
4. *evolved*, *reckless*

For each of these conditions I repeated the experiments seven timse to get an idea of how stable results are. The dependent variables were tracked per generation, but the resulting suite after 50 generations was the focus during evaluation.

6.2.2 Hypotheses

Because the genetic algorithm is configured to prefer tests with high distance of the vehicle to the lane centre, it is expected that, over time, the genetic algorithm produces a test population that cause more out-of-bounds episodes than a randomly generated one. This should be true regardless of driver aggression.

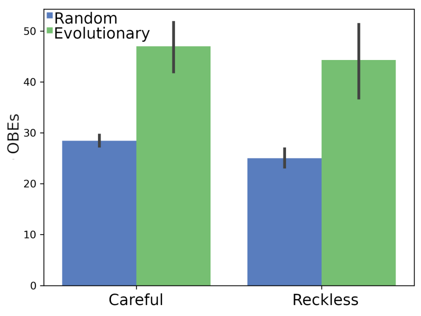
Hypothesis 1 (H1)**.** The evolved conditions produce test suites that cause more Out-of-Bounds Episodes than random conditions for both careful and reckless drivers.

Simulating the 3D environment and physics involved in driving is a very expensive operation, much more so than the geometrical computations involved in road generation or evolution. It is expected that most of the experiment time is spent in executing tests.

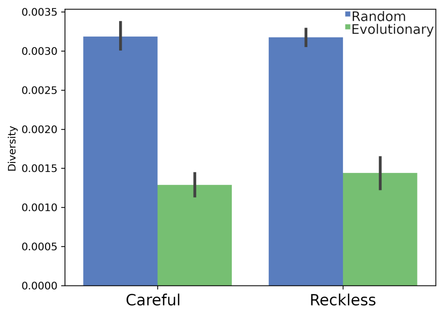
Hypothesis 2 (H2)**.** More than 90% of experiment time is spent executing tests.

The genetic algorithm remixes existing tests which effectively limits suite diversity compared to random generation, which gets the chance to include a completely fresh set of segment combinations in each iteration.

Hypothesis 3 (H3)**.** Suite diversity is lower in the *evolved* conditions than in *random* conditions.



(a) OBE counts



(b) Diversity

Figure 19: This figure illustrates OBE counts and diversity for both *random* and *evolutionary* conditions and each AI aggression setting. The superiority of evolutionary generation with regards to OBE count is clearly shown. Diversity, however, is lacking in the evolutionary settings.

6.2.3 Results

With regards to out-of-bounds episodes, the $evolved, careful$ condition arrived at an average of 47 ($SD = 6.52$) OBEs, whereas $random, careful$ had 28.43 ($SD = 1.5$) OBEs on average. In the $reckless$ settings, $evolved, reckless$ had an average OBE count of 44.29 ($SD = 10.29$) and $random, reckless$ averaged at 25 ($SD = 2.51$) OBEs.

For diversity, the $evolved, careful$ condition produced suites that had an average diversity of 0.13% ($SD = 0.0002\%$), while the suites from $random, careful$ had a diversity of 0.3% ($SD = 0.0002$) on average. On the other hand, $evolved, reckless$ generated suites with an average diversity of 0.14% ($SD = 0.0003$) and $random, reckless$ had 0.3% ($SD = 0.00015$).

Each condition required an average time of 34.24 hours to complete. Of that time, 33.11 hours were spent executing tests.

6.2.4 Discussion

The results confirmed H1, because in the $careful$ and in the $reckless$ settings, the *evolutionary* conditions had on average more OBEs than the *random* ones, confirmed to be significant ($p < 0.01$). This difference can be seen in Figure 19a. This establishes my GA to be superior to random generation, as it outperforms the best suite random could produce within 50 generations. From this follows that generation in this domain requires a non-trivial approach, mine being one option. Interestingly, the $reckless$ conditions ended up with *fewer* OBE counts. This was due to the aggression of the AI causing the car to go out of bounds further and spending more time there. Longer out of bounds episodes resulted in fewer ones, as the AI was still approaching the goal

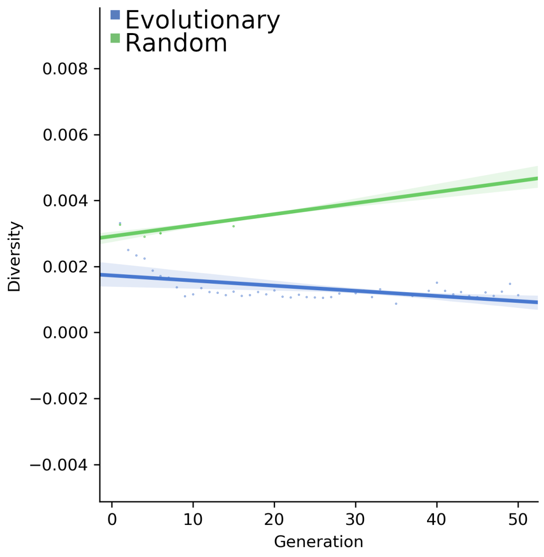
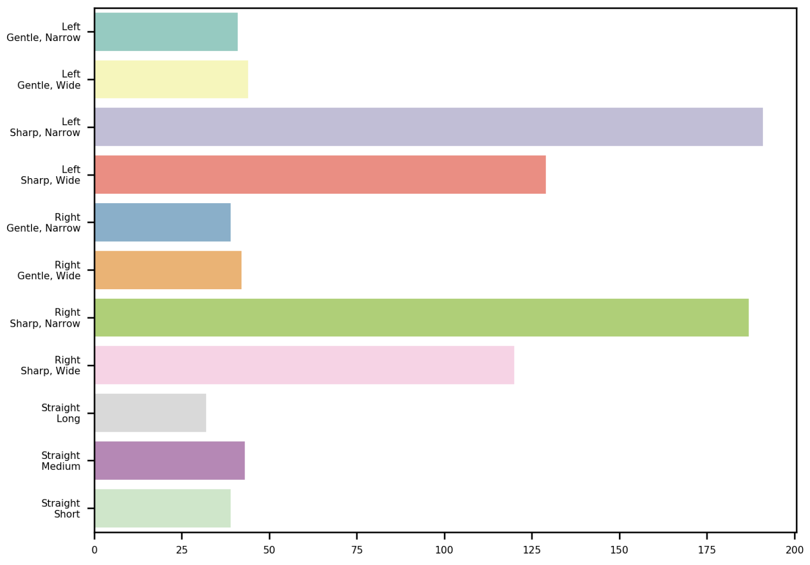


Figure 20: In this figure, an example progression of the suite diversity of both *random* and *evolutionary* generation can be seen. As expected, *random* shows a steady level, while *evolutionary* decays over time as the GA converges to an optimum.

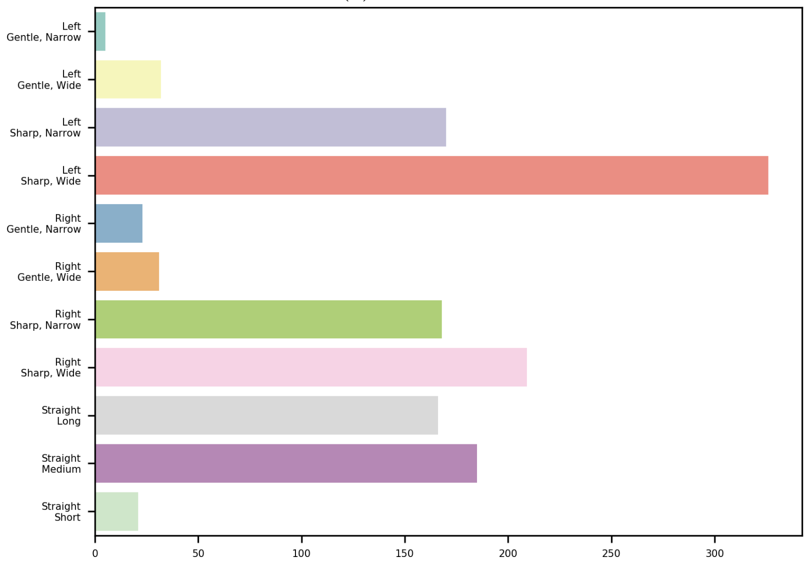
whilst being off track. This effect has implications about the *lanedist*() fitness function that are discussed in Section 7.1.

A look at the split between generation time and execution time also confirms H2. All conditions spent more than 90% of the total generation time executing tests. If one wanted to speed up generation, the simulations should therefore be the focus. With sufficient computing power, execution could be fully parallelised, for example, which would speed up execution of the tests in each generation by a factor of 25 (assuming a population size of 25.) This would *vastly* cut down testing time, going from 34 hours to around 1.5. The kinds of simulations required for testing are inherently expensive to compute, which motivates optimising test generation towards short, yet effective, tests which would reduce execution time. Suite diversity turned out to be problematic in the *evolutionary* conditions; H3 was confirmed with *random* having higher diversity rates than *evolutionary* in both conditions ($p < 0.01$). The comparison is shown in Figure 19b. Diversity was even stagnant in the *evolutionary* conditions, decreasing over the generations. This can be observed in Figure 20. This is an expected result, because the GA prefers fit individuals during selection and therefore converges towards what it knows to be fit. However, this is a problem, as it can blind the suite towards faults not exposed by what's currently considered fit, as discussed in Section 4.6.3. It suggests repetitions of the same flaw among the high OBE count. The random conditions did not suffer from this, as they were able to randomly enumerate possible segment pairs.

As mentioned in Section 4.6, it's possible to infer auxiliary data from a vehicle's trace that does not impact fitness, but sheds some light on where the vehicle went out of bounds. An example of such data can be seen in Figure 21. It shows the segments an OBE was started from that were encountered *during all 50 generations*, not just in the final suite. The distribution of which



(a) Random



(b) Evolutionary

Figure 21: This plot shows the group of road segments the ego-vehicle went out of bounds on for both a random and evolutionary condition. Note that the segment groups here have a lower resolution than the set in D4.23. Turns were grouped into "gentle" ($radius < 45°$) and "sharp" ($radius \geq 45°$), "narrow" ($pivot < 25m$) and "wide" ($pivot \geq 25m$.) Straight segments were grouped into "short" ($length < 100m$), "medium" ($100m < length < 200m$), and "long" ($length \geq 300m$).

segments the car went out of bounds on follows intuition: Sharp and narrow turns are more often the case for OBEs than other segment groups. This chart also confirms earlier intuition from the Diversity Section 4.6.3 that the concept of coverage/diversity needs to focus on more than one segment; the times the vehicle went out of bounds on straight segments are due to the vehicle barely making a turn and then failing to keep to the lane in the segment afterwards. The actual fault happened on the turn, where the vehicle drove in such a way that made it impossible to maintain its position afterwards, but that is not evident from looking at just the segment it broke lane bounds on. It also shows how the GA's distribution of OBE segments differs from random, where the counts are more uniform. The GA, instead, remixed tests that it knew to be effective and ended up with a more lopsided distribution. In conclusion, the GA was established to be superior to random generation with regards to OBE counts, implying the need for more sophisticated generation methods. For diversity, on the other hand, the GA performed worse. Additionally, test execution time was shown to be where most of the generation time is spent.

6.3 Experiment 2: Single- versus Multi-road

Experiment 1 established a baseline of data as to how well evolving roads evokes out of bounds failures in a test subject. Because a single road does not capture the expressive power of my representation or the complexity of real-life road networks, this experiment examined multi- vs. single-road networks. Intersections have played a role in autonomous vehicle incidents before [104, 11, 105]. They also carry with them the benefit of allowing for multiple paths through the network. This can be leveraged for efficiency by re-using one network for multiple tests that cover different paths, but could also reveal problems in the pathing of the AI. For example, in preliminary testing, I observed the AI turning into the wrong lane at intersections and backing out to recover. This exposes Out of Bounds Episodes not possible with a single road. These benefits merit their inclusion and their effect on OBE counts was evaluated in this experiment.

6.3.1 Experiment design

This design follows the format described before. The variation between driver aggression was not performed, however, as its effect was already demonstrated in Experiment 1. This meant all of the following conditions were run with the $careful$ AI setting. The only independent variable in this experiment was:

1. Single vs. multi-road (*single*, *multi*)

To enable multi-road networks, the GA was configured to apply the $merge()$ operator 50% of the time, $join()$ otherwise. This made it evenly split between joining roads and merging networks. The fitness function was the same $lanedist()$ function used in Experiment 1. Because that experiment already gathered data on single-road networks, this experiment only had to be run for the multi-road settings to get comparable data. The conditions were:

1. *single*
2. *multi*

The *multi* condition was run five times to get more stable results.

6.3.2 Hypotheses

The OBE failure count of the resulting suites in the multi-road setting is expected to be higher than in the single-road settings. This is in part due to the GA simply having more options, but also due to how path search works. As described in Section 4.3.3, the algorithm evaluates a random subset of paths and picks the longest one to maximise coverage. While that is not guaranteed to be the longest possible one, it can still traverse more than the single-road settings could, increasing the chances of encountering something the AI can't handle.

Hypothesis 4 (H4)**.** OBE failure counts are higher in $multi, large$ than in $single, large$.

With more options of generating road networks, the GA should also produce suites with higher diversity.

Hypothesis 5 (H4)**.** Suite diversity is higher in $multi, large$ than $single, large$.

6.3.3 Results

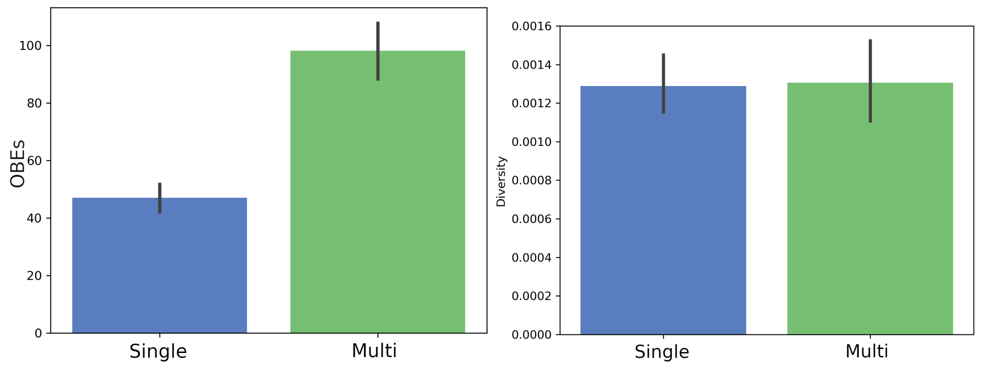
The *single* condition arrived at an average of 47 ($SD = 6.52$) OBEs, whereas *multi* had 98.2 ($SD = 11.16$) OBEs on average. For diversity, the *single* condition produced suites that had an average diversity of 0.13% ($SD = 0.0002\%$) and *multi* had an average diversity of 0.13% ($SD = 0.00022$.) The data for *single* was taken from Experiment 1, but reported here again for readability.

Total generation time, including test execution, was on average 11.2 hours in the $multi, small$ condition. The $multi, large$ condition had an average total generation time of 20.17 hours, on the other hand.

6.3.4 Discussion

I was able to confirm H4. The OBE counts were significantly higher in the *multi* condition than in the *single* condition with $98.2 > 47$, ($p < 0.01$). This difference is visualised in Figure 22a. With regards to diversity, the multi-road condition had no significant improvement. On average, multi-road had a diversity of 0.13%, which is insignificantly more than the 0.12% of the single-road setting ($p > 0.05$.) The diversity comparison can be seen in Figure 22b. The lack of improvement in diversity was likely due to the fact that, despite having had multiple roads per test, these networks were composed of randomly generated single roads; merging various single-road networks into multi-road networks would not increase diversity much. The resulting networks are still made up of the same roads. Intersections between segments would be the main improvement to diversity, which explains the tiny increase multi-road diversity had on average over single-road.

In conclusion, having allowed the GA to use multiple roads more than doubled the final OBE counts. Diversity remained unchanged.



(a) OBE counts (b) Diversity

Figure 22: This figure shows a comparison of OBE counts and diversity in the final suites of single- and multi-road conditions. The superiority of multi-road tests in producing OBE counts is clearly illustrated, while diversity only shows a small improvement for multi-road. Note: the $1km \times 1km$ setting was never performed for single-road networks, so there is no data on it.

6.4 Experiment 3: Increasing diversity

Experiments 1 and 2 have shown the GA, configured for single- and multi-road, to be suffering from low and even stagnant diversity. As fit tests emerge, the GA seemingly gets stuck in creating tests similar to them, failing to cover many of the possible segment pairs. To prevent this, a way of encouraging the GA to create more diverse test suites was needed. Because mates for the next generation with higher fitness get preference during selection, the fitness function was extended to weigh tests according to their relative uniqueness within the suite. What uniqueness means needs to be defined first:

Definition 6.1 (D6.1)**.** The similarity of two tests T_1, T_2 is defined as the Jaccard Index between segment constellations shared and segment constellations in both tests:

$$similarity(T_1, T_2) = \frac{|C_{T_1} \cap C_{T_2}|}{|C_{T_1} \cup C_{T_2}|}$$

Where C_{T_1}, C_{T_2} are the segment constellations covered in T_1 and T_2.

The lower the similarity, the more unique a test is, of course. Uniqueness of a test within a suite is derived from its similarity to the other tests:

Definition 6.2 (D6.2)**.** The uniqueness of a test T_i in a suite $S = (T_1, ..., T_n)$ is the average of the inverted similarity to every other test in S:

$$uniq(T_i, S) = \frac{\sum_{T_j \in S} 1 - similarity(T_i, T_j)}{|S| - 1}$$

This function serves as a weight for the $lanedist()$ fitness used so far. A test that makes a vehicle move away from the lane centre by x metres, but is very similar to other tests, will have that fitness lowered by multiplying it with its uniqueness:

Definition 6.3 (D6.3)**.** Unique fitness of a test T_i of a suite $S = (T_1, ..., T_n)$ with execution trace v is defined as the product between the test's uniqueness and its lane distance fitness:

$$uniqlanedist(T_i, S) = uniq(T_i, S) \cdot lanedist(T_i, v)$$

With this, the most fit but also unique tests are selected to produce the next generation.

6.4.1 Experiment design

To measure the impact of the $uniqlanedist()$ fitness function on the evolution, the multi-road conditions from Experiment 2 were repeated using the new fitness function. Given the fact that smaller networks are traversed more quickly and the confirmation of H2 showing execution time is the biggest factor in overall suite generation time, I wanted to look into the effect of boundary sizes on OBE count. Depending on how effective smaller networks are in causing OBEs, they could be a faster alternative. Therefore, in addition to the $2km \times 2km$ boundary size used in the experiments so far, a $small = 1km \times 1km$ condition was run. The $2km \times 2km$ is from here on called *large*. All conditions ran with the *multi* configuration for the GA from Experiment 2. The main conditions therefore are:

1. $lanedist, small$
2. $lanedist, large$
3. $uniqlanedist, small$
4. $uniqlanedist, large$

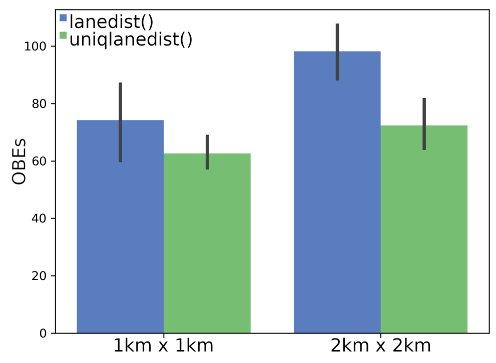
6.4.2 Hypotheses

For the variation in boundary size, I expect OBE failures and suite diversity to scale with the amount of allotted space. How much a suite can actually cover, and how many out-of-bounds-causing segments it can contain, is limited by the space of each test. The *small* condition has half the dimensions of the *large* condition, its OBE counts and diversity should therefore be lower.

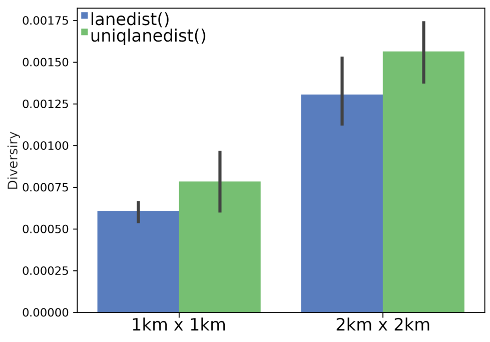
Hypothesis 6 (H6)**.** OBE failures are higher in $multi, large$ than in $multi, small$.

Hypothesis 7 (H7)**.** Suite diversity is higher in $multi, large$ than in $multi, small$.

Because the tests are shorter, the smaller boundary size should lead to faster executions and, by extension, lower total generation times.



(a) OBE counts



(b) Diversity

Figure 23: This figure shows that in both boundary configurations, running the GA with *uniqlanedist*() as the fitness function changed little. Average suite diversity was higher and OBE counts lower in the *uniqlanedist* conditions, but that difference was not confirmed to be significant.

Hypothesis 8 (H8)**.** Total generation time is lower in the *multi*, *small* than in the *multi*, *large* condition.

With *uniqlanedist*() specifically rewarding uniqueness of tests, the expectation is that suite diversity will be higher in the *uniqlanedist* conditions than in the *lanedist* conditions.

Hypothesis 9 (H9)**.** Suite diversity will be higher in the *uniqlanedist* conditions than in the *lanedist* conditions.

Rewarding uniqueness should also solve the issue of diversity actually stagnating over time in the previous experiments. Meaning, not only should it be higher, but it should also not decrease over time:

Hypothesis 10 (H10)**.** Growth of suite diversity over the generations should be positive in the *uniqlanedist* conditions.

The focus on diverse suites should reveal more flaws in the vehicle. Meaning, OOB failure counts should be higher in the respective conditions:

Hypothesis 11 (H11)**.** OBE counts should be higher in the *uniqlanedist* conditions than in the *lanedist* conditions.

6.4.3 Results

The *lanedist*, *small* condition caused an OBE count of 74.2 ($SD = 15.00$) and *uniqlanedist*, *small* had an average OBE count of 62.6 ($SD = 6.18$). For the *large* boundary sizes, *lanedist*, *large*

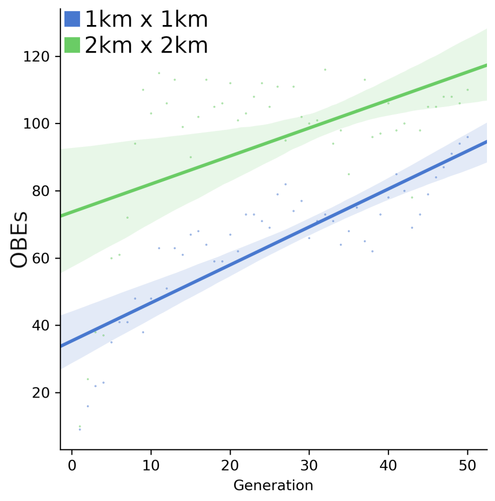


Figure 24: This figure shows suite OBEs per generation for both *small* and *large* sizes. Growth shows a quick rise in OBEs for the first 15 generations and smaller increases afterwards.

had an average OBE count of 98.2 ($SD = 11.16$) and $uniqlanedist, large$ had, on average, 72.4 ($SD = 9.85$) OBEs.

With regards to diversity, the $lanedist, small$ produced suites with an average diversity of 0.06% ($SD = 0.00006$) and $uniqlanedist, small$ generated suites with 0.07% ($SD = 0.0002$) average diversity. In the larger settings, $lanedist, large$ had an average diversity of 0.13% ($SD = 0.00022$) In the $uniqlanedist, large$ condition, the final suite diversity was, on average, 0.16% ($SD = 0.0002$).

6.4.4 Discussion

As expected, both H6 and H7 were confirmed. The difference between *small* and *large* in terms of OBE count and diversity can be seen in Figure 22. It's natural that, with less space, there are fewer opportunities to make the vehicle break lane bounds due to the physical limit of how much road the GA can place alone. However, it is noteworthy that the average OBE count in the *small* condition is, despite having four times less space, only not a fourth of the *large* condition's OBE count. Rather, it's around $^3/_4$, since the average for *small* was 74.2 while the average for *large* was 98.2. Diversity, on the other hand, showed a much more straighforward relationship between boundary size and resulting diversity: Reducing the boundary size by half also roughly halved the diversity, with *small* having an average diversity of 0.06% and *large* an average of 0.13%. This follows the intuition of there simply being a limit as to how much the GA can feasibly place within the boundary and coverage/diversity scaling accordingly.

The total generation time was also reduced in the *small* condition, only slightly more than half of *large*. This also shows a more straightforward relation between boundary size and execution

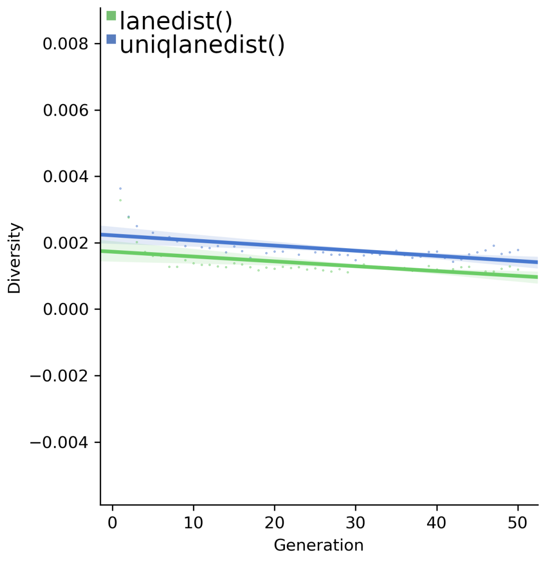


Figure 25: Here, the lack of impact *uniqlanedist*() had on growth is made clear by the respective slopes being almost parallel.

time, similar to diversity. However, given the still-high OBE count of *small*, this suggests small tests to be more efficient, producing more OBEs for the allotted space. Note that the average time even for $2km \times 2km$ conditions was reduced from around 34 hours to 20.17. This is due to optimisations in the generation code implemented between Experiments 1 & 2. While that makes generation times betwen Experiment 1 & 2 incomparable, they are still comparable within Experiment 2 and the fact that the *small* conditions reach 50 generations quicker than the *large* ones still holds. This confirmed H8.

Looking at how OBE counts grow over the 50 generations shows that, in both the *small* and *large* settings, the OBE counts grow very quickly over the first 15 generations and then plateau and show much smaller increases. This pattern can be seen in Figure 24. Since road networks are constrained to a boundary polygon, this pattern suggests that the GA is quick expand roads until it hits the physical limit of how many roads can fit within the boundary space and, from then on, mainly refines those tests for higher OBEs. Depending on the requirements, the testers might therefore reach effective test suites with around a third of the test budget I used.

The results above did not confirm H9. While the average diversity values for the *uniqlanedist* conditions were slightly higher, this difference was not determined to be significant ($p > 0.05$). This can be seen in Figure 23b. As such, it was not possible to conclude *uniqlanedist* to lead to higher suite diversity. A likely explanation for this is how large the set C is and how little variation of diversity values is actually possible within the boundary (see Experiment 2 to get an idea.) This can easily lead to the tiny difference in uniqueness of a test being out-balanced by a larger change in the distance the vehicle went out of lane. Essentially, the set C used to measure uniqueness is of such a high resolution that variations between tests become small to the point of irrelevance. Future work can gather more data to reliably conclude whether H9 is true or not.

It was also not observed that diversity over generations grows positively in the *uniqlanedist*() conditions. See Figure 25 for the diversity progression of an example run. Therefore, H10 was also not confirmed, likely for the same reason as H9.

For OBE counts, the average results for *uniqlanedist*() were lower than for *lanedist*(), but this difference was not statistically significant ($p > 0.05$). The goal of eliminating repeats inherently reduces the total OBE count, which would explain the lower final results, but with no significance to the difference, there's not much more that can be inferred.

To conclude: Factoring in the uniqueness of a test did, while increasing the average diversity slightly, not introduce significant change from Experiment 2. Varying boundary size, however, has revealed that smaller tests can reach disproportionately many OBEs.

6.5 Experiment 4: Mutations

Another attempt at increasing diversity was done by increasing the mutation rate of the GA. The prior experiments all ran with a mutation probability of 50%; for half the produced offspring, the GA attempted a random mutation as described in Section 4.4.3. In this experiment, the probability was changed to 100%. This was done with the expectation of improving upon the ineffective *uniqlanedist*() by raising the impact of a completely random element of the GA on the suites. This was motivated by the *random* generation having had the highest diversity so far.

6.5.1 Experiment design

With the impact of boundary sizes sufficiently demonstrated, this experiment was only run with the *large* boundary sizes from the previous experiments. As reference for comparison, the *lanedist* condition from Experiment 2 was used, as Experiment 3 did not improve upon it in a noteworthy way. The goal remains the same from Experiment 3: improving suite diversity in comparison to a GA using *lanedist*(). In Experiment 3, *lanedist* ran with a mutation rate of $half = 50\%$, while a new condition ran with $all = 100\%$. This condition also used *uniqlanedist*() to give preference to more unique tests. This was the only independent variable of the experiment. The conditions were therefore:

1. $lanedist, half$
2. $uniqlanedist, all$

These were run five times for more stable results.

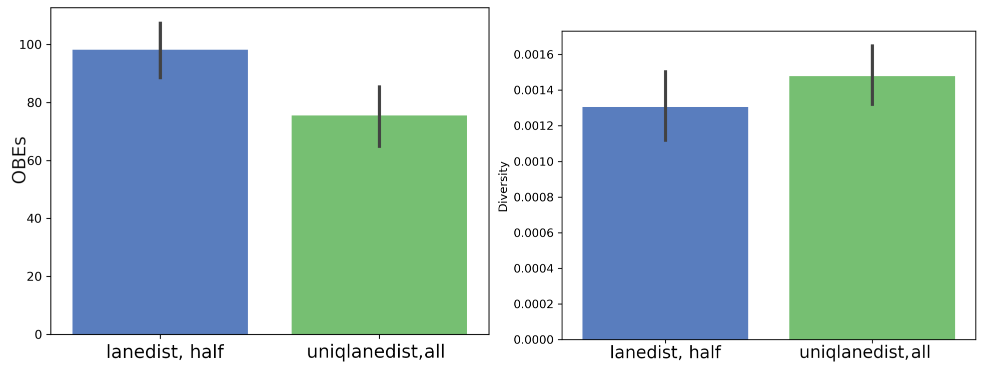
6.5.2 Hypotheses

Increasing randomness during generation should also increase diversity:

Hypothesis 12 (H12)**.** Suite diversity will be higher in $uniqlanedist, all$ than in $lanedist, half$.

With higher diversity, the suite should expose more OBEs:

Hypothesis 13 (H13)**.** OBE count in the $uniqlanedist, all$ condition will be higher.



(a) OBE counts (b) Diversity

Figure 26: In this figure, it can be seen that forcing the GA to always go for mutations did not affect the results with regards to OBE count or diversity significantly.

6.5.3 Results

The *uniqlanedist, all* condition produced test suites that had 75.57 ($SD = 14.20$) OBEs and a diversity of 0.0015 ($SD = 0.0002$) on average.

6.5.4 Discussion

With the results, was not possible to confirm that maximising the mutation rate affected final suite diversity or OBEs. H12 and H13 therefore remain unconfirmed. Figure 26 illustrates the data obtained. The slight differences were not statistically significant ($p > 0.05$). Future work might reveal there to be significant change, as the error margins on the data hint at.

I therefore conclude the mutation rate to be an ineffective way of increasing diversity.

7 Conclusions

My work has built on current research in procedural content generation, road specification, search-based test generation, and driving simulation to define an evolutionary method for generating test scenarios which challenge an autonomous vehicle's lane keeping capabilities. In my evaluation, focused on flat and gapless road networks with a lane for each direction. Despite this focus, I was able to confirm even this basic domain requires non-trivial generation methods was confirmed in a baseline experiment which established my approach to be superior to a method like random generation. Future research needs to look into which methods perform better or worse compared to the one outlined in this thesis.

I was able to generate test suites that effectively evoke lane keeping failures in an autonomous vehicle AI using a generation method that reacts to the vehicle's particular weaknesses. This was achieved within realistic simulations that incur little cost on hardware or danger to people, compared to N-FOT studies. The highest cost has been in time, as generation during experiments took a while, but I have shown various methods of reducing that time, such as varying boundary sizes, limiting the generational budget, and parallelising test executions. With these, my approach is a quick and effective way to find faults within an autonomous vehicle AI. To achieve this, I had to solve problems in the fields of procedural road generation and simulation testing. As the field of simulation testing and automatic test generation for autonomous vehicles evolves, my work serves as reference for solutions to problems in content generation and simulation testing other works face. Finally, the expressive power of my road network model is much larger than what I could feasibly cover in my evaluation. My work therefore lays out a healthy basis for future work in this field.

Final suite diversity is where my approach was lacking: Diversity was lower in the evolved suites than in the random ones. Attempts to increase it were ineffective. The low suite diversity makes it uncertain how many of the failures are actually unique, and not caused by a fault in the vehicle's AI already covered by another test. I would also conclude the diversity metric used to be insufficient. The highest diversity observed were around 0.3%, which was the combination of 25 $2km \times 2km$ networks. This is due to the set of possible segment pairs C being fairly big. On its own, this is not a problem, but when using coverage of that set as a weight, it means in-/exclusion of a few new segment pairs has a very small effect on the overall weight of a test. Combined with a fitness function like $lanedist()$, this means a small change in weight was easily compensated for by tests that cause higher lane distances. This suggests lowering the resolution of segment grouping to have fewer pairs in C and normalising lane distance to be in the same range as the weight: $[0, 1]$. For $lanedist()$, this could be done by defining 0 as being on the lane centre, 1 as being out of bounds, and any distance inbetween to $(0, 1)$. Future work should look into the impact of this.

Like with other simulation-based approaches, one of the biggest problems has been the execution time of tests. Running a single condition of an experiment took almost two days per run, with more than 90% spent executing tests in the simulator. This is despite the simulator being configured to run faster than real time. Experiments were run on a consumer PC (3.10 Ghz i7-4770S Quad Core, 16 GiB RAM, GeForce GTX 970) that was more than powerful enough to handle four instances of the simulator at the same time. Yet, experiments took so long that gathering a significant amount of samples became unfeasible (at 34h, $n = 30$ would have required almost one and a half months.) However, test execution can also be fully parallelised. With sufficient computing power, it can be reduced by almost a factor of n, where n is the population size of the GA. Besides accelerating actual executions, it could also be possible to intelligently eliminate unnecessary executions or employ surrogate testing to reduce generation time.

It's noteworthy that the OBE failure rate, while increasing steadily during evolution, also varies a lot from generation to generation. This is due to some actually fit tests not being carried over into the next generation, either because the randomised tournament selection didn't settle on them, or the randomised operations done on them ended up creating tests that were worse. One way to alleviate this is increasing the elitism of the GA, which would make sure a certain amount of good tests always get taken into the next generation. Another way is to switch the GA from a generational model, where the population is replaced by the evolved one in each generation, to a “steady state” one, that maintains a population of the n most fit individuals. With that setting, offspring is inserted into the same population, evaluated for fitness, and then only the best n are kept. This would make sure any offspring that’s worse is not kept. Future work should examine how that affects final OBE counts. These methods would negatively impact diversity, as the GA would focus on what it considers fit even quicker. However, this might be addressed by employing a multi-objective search such as NSGA-II [106] that maximises for both OBE counts and diversity.

7.1 Limitations

This method has so far been only evaluated on a perfect knowledge AI, which does not represent autonomous vehicle AIs used in the industry [65]. As discussed, this increases internal validity but is a significant hit to external validity. While the proposed approach treats the subject as a black box and is applicable to any test subject whose lane keeping behaviour needs to be tested, this still casts doubt on the merits of this approach to autonomous vehicle AIs from the industry. Lack of availability and portability is one reason no actual autonomous vehicle AI was considered during evaluation, but the other reason hints at another limitation of this approach: Adapting an AI to be used with the simulator was met with extreme difficulty, which needs to be overcome before testing like this can be applied to autonomous vehicle software.

As hinted in the evaluation, the visual components — besides its shape, of course — are not part of the representation. This blinds the approach from faults caused by varying visual aspects in AIs whose lane-keeping behaviour is determined by visual processing. What the lane boundaries look like, the texture of the road, time of day and corresponding environmental brightness, and so on, are important when a vehicle processes a camera feed to identify drive-able space, but is not captured in my representation. Works like [59] include these specifically to also test the image processing algorithm’s lane recognition.

Requiring the vehicle to follow a fixed path through a network also has its limitations. The requirement was in service of analysis: One path makes it clear how far away the vehicle is from following it and what the driving task actually requires from the vehicle (i.e. coverage.) However, this limits the possible subjects to vehicles who can be given a path to follow, as opposed to merely a goal point to reach, and prevents testing path search of the vehicle itself. This could even affect lane-keeping behaviour: A vehicle AI could be programmed to avoid paths with turns it knows it’s unlikely to successfully navigate and choose paths accordingly. This would in turn lower the OBE failure rate. Future work should look into only defining a start and goal point for the ego-vehicle and analyse the path it chose, instead.

The low sample count for experiments is a threat to validity. While I was still able to obtain low p-values for most comparisons, the result of Experiment 2 in particular suffer from lack of data. This experiment was concluded unsuccessful, as significant increase in diversity was not observed. As Figure 23b shows, the average diversity in the suites generated with $uniqlanedist()$ was higher, but the margin for error fairly large, making conclusions unreliable. If more data was gathered, this might change to confirm or confidently rule out $uniqlanedist()$ as a way of increasing diversity.

8 Future Work

While the model and algorithms I defined in my work are powerful, they do not capture the complexity of real-world road networks and therefore can never cover all of the situations an autonomous vehicle AI will face. Futhermore, my evaluation also examined only a subset of my model's the expressive power. Future work should look into both expanding my approach to allow for more complex situations, but also extend evaluation towards other areas of the model I defined or changes that the evaluation suggested.

8.1 Different Diversity Metric

As the Conclusion Section 7 discussed, the diversity metric used was shown to be flawed. Besides the size of the set of possible pairs of segments C, future work should also look into a diversity metric that is more suitable for intersections. In my work, an intersection counts as one way to combine segments. Intuitively, however, there's a difference between an intersection that crosses at the apex of a turn and an intersection that crosses a turn at the start or end of it, when the shape of the road is more straight. As it stands, my concept of diversity does not address this. The increase in OBE rates observed after intersections were introduced in Experiment 2 motivates a diversity metric that is more tailored towards intersections, as they have shown to be the cause of many OBEs and, without a meaningful concept of diversity, it's uncertain how unique these are.

8.2 Varying Height

My road networks are fully flat. All formalisations given in Section 4 rely on two-dimensional coordinates. These definitions could be extended to include a z-coordinate to express height. Road segments could then transition from, for example, a height of 1 metre to 3 metres. With regards to lane keeping, this might expose new faults, such as the vehicle miscalculating the acceleration needed to overcome a slope, underestimating the increased speed from going downhill, or even driving too quickly up a slope and losing traction from an accidental jump. However, this change would require some new constraints. With the option to ascend off ground, roads could also start overlapping without actually intersecting — similar to a bridge crossing over another road. These would not be intersections and would have to impose a required minimum height that still allows the vehicle to pass under the bridge for the test to be solvable.

8.3 Varying Lane Width

Roads in the real world can vary in width, but my evaluation worked on lanes that all had the same width. If the road segment generation algorithm and the GA were altered to allow for variations in lane width, faults specific to the ego-vehicle being unable to cope with such changes could be exposed. For example, Schuldt et al. [42] focused their evaluation on a scenario where the ego-vehicle had to navigate a road that got more narrow due to a construction site blocking parts of it. Failures in such a setting are not detected by my algorithm as it is and future work could look into expanding it to cover them.

8.4 Multiple Lanes

The model I described is built to support multiple lanes. My evaluation focused on the case of one lane on each side. Future work could examine how my approach scales to multiple lanes. This would require altering the definition of the vehicle trace to not just focus on one lane, but measure the ego-vehicle's distance to the lane centre of the lane it is currently driving on. In turn, it would allow for a new manoeuvre and source of failure: lane switching. With multiple lanes, the ego-vehicle is given the option to switch lanes as it sees fit. This opens it up to failure during that and might increase OBE counts.

8.5 Obstacles on the Road

Having multiple lanes would also open up the possibility of adding obstacles such as other vehicles to the lanes. With a single lane, this is not really possible, as avoiding said obstacle would force the vehicle to leave the lane and thus force it to fail. With multiple lanes, the networks could place obstacles in a way that forces a lane switching maneuvre and possibly reveal failure to detect the obstacle at all or failures during lane switching. However, test generation would need to ensure obstacles are not placed in a way that makes them impossible to avoid, as multiple obstacles placed next to each other on multiple lanes can end up blocking any way ahead for the ego-vehicle.

8.6 Let Ego-Vehicle Pick a Path

My current representation forces the vehicle down a path that is fixed for each test. This could be changed to merely give the vehicle a goal position to drive to and let it perform its own path search. This would allow evaluating how well the vehicle's pathing algorithm deals with complicated situations. The evaluation of the vehicle trace would need to be adapted to account for this, however. It could no longer measure the distance to the lane centre along the target path. Instead, the path the ego-vehicle took throughout the network would have to be inferred from its positions and which segments it drove on and the distance to the lane centres calculated based on those.

References

[1] P. Bansal and K. M. Kockelman, “Forecasting Americans’ long-term adoption of connected and autonomous vehicle technologies,” *Transportation Research Part A: Policy and Practice*, vol. 95, pp. 49–63, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965856415300628

[2] SAE International, “Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.”

[3] Waymo, “Waymo.” [Online]. Available: https://waymo.com/

[4] “Autopilot.” [Online]. Available: https://www.tesla.com/autopilot

[5] UberATG, “Self-driving cars return to Pittsburgh roads in manual mode,” Jul. 2018. [Online]. Available: https://medium.com/@UberATG/self-driving-cars-return-to-pittsburgh-roads-in-manual-mode-f83e506a04b9

[6] J. M. Gitlin, “Volvo’s autonomous Drive Me research project gets underway,” Jan. 2017. [Online]. Available: https://arstechnica.com/cars/2017/01/volvos-autonomous-drive-me-research-project-gets-underway/

[7] F. Lambert, “A new Tesla Autopilot update is ‘in final testing phase’, says Elon Musk,” Feb. 2018. [Online]. Available: https://electrek.co/2018/02/02/tesla-autopilot-new-update-elon-musk/

[8] M. Bertoncello and D. Wee, “Ten ways autonomous driving could redefine the automotive world | McKinsey & Company.” [Online]. Available: http://www.mckinsey.com/industries/automotive-and-assembly/our-insights/ten-ways-autonomous-driving-could-redefine-the-automotive-world

[9] “Autonomous Vehicle Disengagement Reports 2017.” [Online]. Available: https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2017

[10] N. Zhou, “Volvo admits its self-driving cars are confused by kangaroos,” *The Guardian*, Jul. 2017. [Online]. Available: http://www.theguardian.com/technology/2017/jul/01/volvo-admits-its-self-driving-cars-are-confused-by-kangaroos

[11] A. Davies, “Google’s Self-Driving Car Caused Its First Crash,” *Wired*, Feb. 2016. [Online]. Available: https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/

[12] M. Wehner, “Video shows Tesla Model S slamming into a wall while driving on Autopilot,” Mar. 2017. [Online]. Available: https://bgr.com/2017/03/02/tesla-crash-video-texas/

[13] O. Solon, “Tesla that crashed into police car was in ’autopilot’ mode, California official says,” *The Guardian*, May 2018. [Online]. Available: http://www.theguardian.com/technology/2018/may/29/tesla-crash-autopilot-california-police-car

[14] E. Shilling, “Video Appears To Show Tesla Autopilot Veering Toward Divider At Site Of Deadly Crash.” [Online]. Available: https://jalopnik.com/video-appears-to-show-tesla-autopilot-veering-toward-di-1825016336

[15] CNBC, “Uber fatal crash: Self-driving SUV saw pedestrian, didn’t brake: feds,” May 2018. [Online]. Available: https://www.cnbc.com/2018/05/24/ubers-self-driving-suv-saw-the-pedestrian-in-fatal-accident-but-didnt-brake-officials-say.html

[16] S. Levin, “Tesla fatal crash: ’autopilot’ mode sped up car before driver killed, report finds,” *The Guardian*, Jun. 2018. [Online]. Available: http://www.theguardian.com/technology/2018/jun/07/tesla-fatal-crash-silicon-valley-autopilot-mode-report

[17] Y. Tian, K. Pei, S. Jana, and B. Ray, “DeepTest: Automated testing of deep-neural-network-driven autonomous cars,” *arXiv preprint arXiv:1708.08559*, 2017.

[18] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, “DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing,” *arXiv:1802.02295 [cs]*, Feb. 2018, arXiv: 1802.02295. [Online]. Available: http://arxiv.org/abs/1802.02295

[19] L. Li, W. L. Huang, Y. Liu, N. N. Zheng, and F. Y. Wang, “Intelligence Testing for Autonomous Vehicles: A New Approach,” *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, Jun. 2016, alessio.

[20] S. Khastgir, S. Birrell, G. Dhadyalla, and P. Jennings, “Identifying a gap in existing validation methodologies for intelligent automotive systems: Introducing the 3xd simulator,” in *2015 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2015, pp. 648–653, alessio.

[21] P. Koopman and M. Wagner, “Challenges in Autonomous Vehicle Testing and Validation,” *SAE Int. J. Trans. Safety*, vol. 4, no. 1, pp. 15–24, Apr. 2016, alessio. [Online]. Available: http://papers.sae.org/2016-01-0128/

[22] M. Mauritz, F. Howar, and A. Rausch, “Assuring the Safety of Advanced Driver Assistance Systems Through a Combination of Simulation and Runtime Monitoring,” in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, ser. Lecture Notes in Computer Science. Springer, Cham, Oct. 2016, pp. 672–687, alessio Marc. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-47169-3_52

[23] C. Sippl, F. Bock, D. Wittmann, H. Altinger, and R. German, “From Simulation Data to Test Cases for Fully Automated Driving and ADAS,” in *Testing Software and Systems*, ser. Lecture Notes in Computer Science. Springer, Cham, Oct. 2016, pp. 191–206, alessio. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-47443-4_12

[24] M. Aeberhard, S. Rauch, M. Bahram, G. Tanzmeister, J. Thomas, Y. Pilat, F. Homm, W. Huber, and N. Kaempchen, “Experience, Results and Lessons Learned from Automated Driving on Germany’s Highways,” *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 42–57, 2015.

[25] E. Royer, F. Marmoiton, S. Alizon, D. Ramadasan, M. Slade, A. Nizard, M. Dhome, B. Thuilot, and F. Bonjean, “Lessons learned after more than 1000 km in an autonomous shuttle guided by vision,” in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2016, pp. 2248–2253.

[26] M. Bertozzi, A. Broggi, A. Coati, and R. I. Fedriga, “A 13,000 km Intercontinental Trip with Driverless Vehicles: The VIAC Experiment,” *IEEE Intelligent Transportation Systems Magazine*, vol. 5, no. 1, pp. 28–41, 2013.

[27] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?” *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, Dec. 2016, alessio. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965856416302129

[28] S. Masuda, “Software Testing Design Techniques Used in Automated Vehicle Simulations,” in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Mar. 2017, pp. 300–303, alessio.

[29] D. Gruyer, S. Glaser, B. Vanholme, and B. Monnier, “Simulation of automatic vehicle speed control by transponder-equipped infrastructure.” IEEE, Oct. 2009, pp. 628–633. [Online]. Available: http://ieeexplore.ieee.org/document/5399281/

[30] P. Minnerup and A. Knoll, “Testing Automated Vehicles Against Actuator Inaccuracies in a Large State Space,” *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 38–43, Jan. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896316308825

[31] J.-H. Park and Y.-W. Tai, “A simulation based method for vehicle motion prediction,” *Computer Vision and Image Understanding*, vol. 136, pp. 79–91, Jul. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314215000508

[32] M. Althoff, M. Koschi, and S. Manzinger, “CommonRoad: Composable benchmarks for motion planning on roads,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 719–726, alessio.

[33] R. Molenaar, A. v. Bilsen, R. v. d. Made, and R. d. Vries, “Full spectrum camera simulation for reliable virtual development and validation of ADAS and automated driving applications,” in *2015 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2015, pp. 47–52.

[34] M. Nentwig and M. Stamminger, “Hardware-in-the-loop testing of computer vision based driver assistance systems,” in *2011 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2011, pp. 339–344, alessio.

[35] D. Hospach, S. Mueller, O. Bringmann, J. Gerlach, and W. Rosenstiel, “Simulation and evaluation of sensor characteristics in vision based advanced driver assistance systems,” in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2014, pp. 2610–2615.

[36] M. R. Zofka, R. Kohlhaas, T. Schamm, and J. M. Zöllner, “Semivirtual simulations for the evaluation of vision-based ADAS,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, Jun. 2014, pp. 121–126, alessio.

[37] J. V. Casas, A. Torday, and A. Gerodimos, “Combining Mesoscopic and Microscopic Simulation in an Integrated Environment as a Hybrid Solution,” *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 3, pp. 25–33, 2010.

[38] C. Berger, M. Chaudron, R. Heldal, O. Landsiedel, and E. M. Schiller, “Model-based, composable simulation for the development of autonomous miniature vehicles,” in *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium.* Society for Computer Simulation International, 2013, p. 17, alessio. [Online]. Available: http://dl.acm.org/citation.cfm?id=2499651

[39] G. Bagschik, T. Menzel, and M. Maurer, "Ontology based Scene Creation for the Development of Automated Vehicles," *arXiv:1704.01006 [cs]*, Mar. 2017, alessio. [Online]. Available: http://arxiv.org/abs/1704.01006

[40] D. Gruyer, O. Orfila, V. Judalet, S. Pechberti, B. Lusetti, and S. Glaser, "Proposal of a virtual and immersive 3d architecture dedicated for prototyping, test and evaluation of eco-driving applications," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2013, pp. 511–518, alessio.

[41] C. Zhang, Y. Liu, D. Zhao, and Y. Su, "RoadView: A traffic scene simulator for autonomous vehicle simulation testing," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2014, pp. 1160–1165, alessio.

[42] F. Schuldt, "Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen, Towards testing of automated driving functions in virtual driving environments," Apr. 2017, marc. [Online]. Available: https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00064747

[43] A. Andrews, M. Abdelgawad, and A. Gario, "Towards world model-based test generation in autonomous systems," in *Model-Driven Engineering and Software Development (MODELSWARD), 2015 3rd International Conference on.* IEEE, 2015, pp. 1–12, alessio. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7323096/

[44] S. Lee, J. Kim, H. Song, and S. Kim, "A 3-dimensional real-time traffic simulator considering the interaction among autonomous and human-driven vehicles," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2014, pp. 1917–1918.

[45] J. Felez, J. Maroto, J. M. Cabanellas, and J. M. Mera, "A full-scale simulation model to reproduce urban traffic in real conditions in driving simulators," *SIMULATION*, vol. 89, no. 9, pp. 1099–1114, Sep. 2013. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0037549713483557

[46] D. Sportillo, A. Paljic, M. Boukhris, P. Fuchs, L. Ojeda, and V. Roussarie, "An Immersive Virtual Reality System for Semi-autonomous Driving Simulation: A Comparison Between Realistic and 6-DoF Controller-based Interaction," in *Proceedings of the 9th International Conference on Computer and Automation Engineering*, ser. ICCAE '17. New York, NY, USA: ACM, 2017, pp. 6–10, alessio. [Online]. Available: http://doi.acm.org/10.1145/3057039.3057079

[47] M. Feilhauer and J. Häring, "A multi-domain simulation approach to validate Advanced Driver Assistance Systems," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2016, pp. 1179–1184.

[48] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving." IEEE, Dec. 2015, pp. 2722–2730, alessio. [Online]. Available: http://ieeexplore.ieee.org/document/7410669/

[49] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?" *arXiv:1610.01983 [cs]*, Oct. 2016, arXiv: 1610.01983. [Online]. Available: http://arxiv.org/abs/1610.01983

[50] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” *arXiv:1711.03938 [cs]*, Nov. 2017, arXiv: 1711.03938. [Online]. Available: http://arxiv.org/abs/1711.03938

[51] Rockstar Games, “Grand Theft Auto V,” 2013. [Online]. Available: https://www.rockstargames.com/V/

[52] E. Makuch, “Rockstar: More than 1,000 people made GTAV.” [Online]. Available: https://www.gamespot.com/articles/rockstar-more-than-1000-people-made-gtav/1100-6415330/

[53] M. Martinez, C. Sitawarin, K. Finch, L. Meincke, A. Yablonski, and A. Kornhauser, “Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars,” *arXiv:1712.01397 [cs]*, Dec. 2017, arXiv: 1712.01397. [Online]. Available: http://arxiv.org/abs/1712.01397

[54] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, “What is Procedural Content Generation?: Mario on the Borderline,” in *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*, ser. PCGames ’11. New York, NY, USA: ACM, 2011, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/2000919.2000922

[55] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*, 1st ed. Springer Publishing Company, Incorporated, 2016.

[56] R. M. Smelik, “A declarative approach to procedural generation of virtual worlds.” Ph.D. dissertation, [s.n.], S.l., 2011.

[57] 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*, vol. 3, no. 3, pp. 245–259, Sep. 2011, alessio.

[58] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, “Testing advanced driver assistance systems using multi-objective search and neural networks,” in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2016, pp. 63–74, alessio.

[59] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, “Learning to Drive in a Day,” *arXiv:1807.00412 [cs, stat]*, Jul. 2018, arXiv: 1807.00412. [Online]. Available: http://arxiv.org/abs/1807.00412

[60] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, “Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on.* IEEE, 2017, pp. 1443–1450, alessio. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7989173/

[61] S. Huang and D. Ramanan, “Expecting the Unexpected: Training Detectors for Unusual Pedestrians with Adversarial Imposters,” 2017, alessio Marc. [Online]. Available: https://pdfs.semanticscholar.org/9e53/78e7b336c89735d3bb15cf67eff96f86d39a.pdf

[62] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, “Procedural Content Generation: Goals, Challenges and Actionable Steps,” in *Artificial and Computational Intelligence in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 61–75. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4336

[63] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press, 1996.

[64] J. Fritsch, T. Kühnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, Oct. 2013, pp. 1693–1700.

[65] F. Falcini, G. Lami, and A. M. Costanza, "Deep Learning in Automotive Software," *IEEE Software*, vol. 34, no. 3, pp. 56–63, May 2017, alessio.

[66] "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org/

[67] BeamNG GmbH, "BeamNG.research – BeamNG." [Online]. Available: https://beamng.gmbh/research/

[68] "OpenDRIVE - Home." [Online]. Available: http://www.opendrive.org/

[69] J.-D. Boissonnat and M. Teillaud, Eds., *Effective Computational Geometry for Curves and Surfaces*, ser. Mathematics and Visualization. Berlin Heidelberg: Springer-Verlag, 2006. [Online]. Available: //www.springer.com/de/book/9783540332589

[70] J. Y. C. Chen and J. E. Thropp, "Review of Low Frame Rate Effects on Human Performance," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 6, pp. 1063–1076, Nov. 2007.

[71] "List of Unity games," Aug. 2018, page Version ID: 853677984. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_Unity_games&oldid=853677984

[72] "List of Unreal Engine games," Aug. 2018, page Version ID: 853716981. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_Unreal_Engine_games&oldid=853716981

[73] "Torque 3d | Products | GarageGames.com." [Online]. Available: http://www.garagegames.com/products/torque-3d

[74] A. Belbachir, "An embedded testbed architecture to evaluate autonomous car driving," *Intel Serv Robotics*, vol. 10, no. 2, pp. 109–119, Apr. 2017. [Online]. Available: https://link.springer.com/article/10.1007/s11370-016-0213-6

[75] M. Althoff and J. M. Dolan, "Online Verification of Automated Road Vehicles Using Reachability Analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, Aug. 2014, alessio.

[76] D. Yadron and D. Tynan, "Tesla driver dies in first fatal crash while using autopilot mode," *The Guardian*, Jun. 2016. [Online]. Available: http://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk

[77] J. Zhou, R. Schmied, A. Sandalek, H. Kokal, and L. del Re, "A Framework for Virtual Testing of ADAS," *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 9, no. 1, Apr. 2016. [Online]. Available: http://papers.sae.org/2016-01-0049/

[78] D. Zhao, X. Huang, H. Peng, H. Lam, and D. J. LeBlanc, "Accelerated Evaluation of Automated Vehicles in Car-Following Maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, no. 99, pp. 1–12, 2017.

[79] D. Zhao and H. Peng, "From the Lab to the Street: Solving the Challenge of Accelerating Automated Vehicle Testing," *arXiv preprint arXiv:1707.04792*, 2017, alessio. [Online]. Available: https://arxiv.org/abs/1707.04792

[80] S. Müller, D. Hospach, O. Bringmann, J. Gerlach, and W. Rosenstiel, "Robustness Evaluation and Improvement for Vision-Based Advanced Driver Assistance Systems," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sep. 2015, pp. 2659–2664, alessio.

[81] A. C. Madrigal, "Inside Waymo's Secret World for Training Self-Driving Cars," *The Atlantic*, Aug. 2017. [Online]. Available: https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/

[82] A. Ajmera, "Audi partners with Israel's autonomous vehicle simulation startup..." *Reuters*, Jun. 2018. [Online]. Available: https://www.reuters.com/article/us-cognata-audi/audi-partners-with-israels-autonomous-vehicle-simulation-startup-cognata-idUSKBN1JM15X

[83] M. Nentwig, M. Miegler, and M. Stamminger, "Concerning the applicability of computer graphics for the evaluation of image processing algorithms," in *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, Jul. 2012, pp. 205–210, marc.

[84] "TORCS," 1997. [Online]. Available: http://torcs.sourceforge.net/

[85] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for Data: Ground Truth from Computer Games," in *Computer Vision – ECCV 2016*, ser. Lecture Notes in Computer Science. Springer, Cham, Oct. 2016, pp. 102–118, alessio. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46475-6_7

[86] Unity, "Unity 3d." [Online]. Available: https://unity3d.com/

[87] "Mapzen · an open, sustainable, and accessible mapping platform," 2013. [Online]. Available: https://mapzen.com/

[88] "Unreal Engine 4," 2014. [Online]. Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4

[89] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," *arXiv:1705.05065 [cs]*, May 2017, arXiv: 1705.05065. [Online]. Available: http://arxiv.org/abs/1705.05065

[90] "deepdrive: The easiest way to experiment with self-driving AI," Jun. 2018, original-date: 2017-11-21T17:28:45Z. [Online]. Available: https://github.com/deepdrive/deepdrive

[91] D. Etherington, "Udacity open sources its self-driving car simulator for anyone to use." [Online]. Available: http://social.techcrunch.com/2017/02/08/udacity-open-sources-its-self-driving-car-simulator-for-anyone-to-use/

[92] BeamNG, "BeamNG.drive." [Online]. Available: https://www.beamng.com/

[93] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural Content Generation for Games: A Survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, pp. 1:1–1:22, Feb. 2013, alessio. [Online]. Available: http://doi.acm.org/10.1145/2422956.2422957

[94] H. Long, "Procedurally generated realistic virtual rural worlds," Thesis, University of Cape Town, 2016. [Online]. Available: https://open.uct.ac.za/handle/11427/20874

[95] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A Survey on Procedural Modelling for Virtual Worlds," *Computer Graphics Forum*, vol. 33, no. 6, pp. 31–50, Sep. 2014, alessio. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/cgf.12276/abstract

[96] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," in *ACM transactions on graphics (TOG)*, vol. 27. ACM, 2008, p. 103, alessio. [Online]. Available: http://dl.acm.org/citation.cfm?id=1360702

[97] Y. I. H. Parish and P. Müller, "Procedural Modeling of Cities," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 301–308. [Online]. Available: http://doi.acm.org/10.1145/383259.383292

[98] L. Cardamone, P. L. Lanzi, and D. Loiacono, "TrackGen: An interactive track generator for TORCS and Speed-Dreams," *Applied Soft Computing*, vol. 28, pp. 550–558, Mar. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494614005705

[99] T. Georgiou and Y. Demiris, "Personalised track design in car racing games," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sep. 2016, pp. 1–8, alessio.

[100] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated Car Racing Championship: Competition Software Manual," *arXiv:1304.1672 [cs]*, Apr. 2013, arXiv: 1304.1672. [Online]. Available: http://arxiv.org/abs/1304.1672

[101] "CommonRoad." [Online]. Available: http://commonroad.gitlab.io/

[102] R. Sedgewick, *Algorithms in C, Part 5: Graph Algorithms, Third Edition*, 3rd ed. Addison-Wesley Professional, 2001.

[103] A. Arcuri and L. Briand, "A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering," *Softw. Test. Verif. Reliab.*, vol. 24, no. 3, pp. 219–250, May 2014. [Online]. Available: http://dx.doi.org/10.1002/stvr.1486

[104] S. Klingelschmitt, F. Damerow, and J. Eggert, "Managing the complexity of inner-city scenes: An efficient situation hypotheses selection scheme," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2015, pp. 1232–1239, alessio.

[105] C. Williams and E. i. C. . A. . a. . tweet_btn(), "Stop lights, sunsets, junctions are tough work for Google's robo-cars." [Online]. Available: https://www.theregister.co.uk/2016/08/24/google_self_driving_car_problems/

[106] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.