Automatically Reconstructing Car Crashes from Police Reports for Testing Self-Driving Cars

Undisclosed Authors

Abstract—**Autonomous driving carries the promise to drastically reduce car accidents; however, recently reported fatal crashes involving self-driving cars make this hard to believe, and demonstrate that software controlling self-driving cars needs to be better tested. This, however, is challenging because it requires producing critical scenarios either in the real world, or in simulations. To better test self-driving cars, we propose AC3R (Automatic Crash Constructor from Crash Report), which creates simulated test scenarios representing critical situations in which human drivers caused accidents, as documented in police reports. By automatically recreating recorded car crashes as accurate simulations, AC3R (1) tames the huge test input space by helping to find relevant test cases, and (2) allows testers to generate a plethora of relevant test cases from the massive historical dataset of recorded car crashes. Our extensive evaluation, consisting of a user study involving 34 participants and state-of-art self-driving car software, shows that AC3R generates accurate simulations in a matter of minutes. Our tests not only exposed several shortcomings in the test subject, but also found a critical bug in its implementation.**

I. Introduction

Autonomous driving is the industry's promise to eliminate 90% of all accidents, and the sales of self-driving cars are expected to hit $87 billion dollars by 2030 [1]. However, crash reports involving self-driving cars and advanced driver-assistance systems (ADAS) [2], some of which were fatal for drivers [3] and pedestrians [4], testify that current autonomous driving cannot be considered as safe as promoted. In many cases, the reason is defective software. For example, in the infamous fatal accidents involving Tesla and Uber self-driving cars, the software respectively interpreted an incoming truck as an overhead road sign [3], and a pedestrian crossing the street as an unknown object *and* a vehicle [5]. This clearly indicates the need for a better approach for testing self-driving cars.

The current practice for testing self-driving cars consists of naturalistic field operational tests, in which self-driving cars are left free to drive in the hope of observing interesting events while not causing accidents. However, this is not only highly impractical, dangerous, and ineffective [6], but also it avoids precisely the most interesting, *critical* testing scenarios. An alternative lies in *virtual tests*, where computer simulations are used to challenge self-driving car software [7]–[9]. While this provides the opportunity to automatically generate tests (e.g., [10]–[13]), the main open challenge is what constitutes good test scenarios, and how to systematically generate them.

In this paper, we address exactly this problem and propose AC3R, a novel approach to efficiently generate relevant test cases for testing self-driving cars by automatically configuring computer simulations. The main insight underlying AC3R is that *real car crashes* represent critical situations that challenge self-driving cars, and would thus be perfectly suited as test scenarios. Unfortunately, detailed sensory data collected by the cars [14] during real crashes is rarely available and detailed enough to enable a full reproduction. Therefore, AC3R adopts a different strategy and extracts the information to recreate the car crashes from the *police reports* which document them.

Police reports are the result of analytical work done by experts, and contain all relevant details about car crashes and their contributing factors. They are organized according to public standard guidelines, cover many types of accidents, and are stored in large, usually publically accessible, databases, which favor diversity of the generated test cases and automation. The technical challenge is that police reports are multi-media documents, hence only partially structured, and written in natural language. Therefore, AC3R combines Natural Language Processing (NLP) techniques with a domain-specific ontology for extracting the relevant information from the police reports, reconstructing the car crash dynamics using kinetics models, and automatically generating the code which re-enacts the car crashes as computer simulations, suitable as test cases.

In detail, this paper makes the following contributions:

- We present AC3R, the first approach for automatically reconstructing car crashes from police reports and using them for testing self-driving cars.
- We evaluate the accuracy of the reconstructed car crashes by means of an empirical user study.
- We evaluate the quality of the generated test cases by testing a state-of-art self-driving car software.

Our extensive evaluation shows that AC3R is efficient and generates simulations that accurately reproduce several types of crashes in minutes; further, the test cases we derived from such simulations exposed several shortcomings and a critical bug in state-of-art self-driving car software. To favor the replication of our studies as well as to promote enable further research on this topic we release AC3R's source code, evaluation data, and instructions to replicate the experiments at: (blinded URL)

II. Automatically Reconstructing and Simulating Real Car Crashes with AC3R

AC3R automatically reconstructs car crashes from police reports as simulations in four main steps (Figure 1):

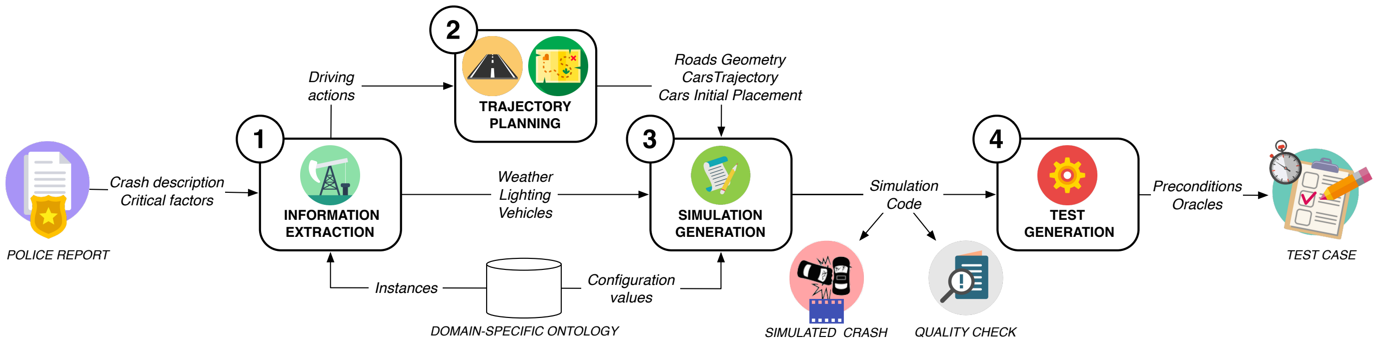

Fig. 1. AC3R Overview. The figure illustrates the main steps which compose the proposed approach: information extraction, trajectory planning, simulation generation, and tests generation.

1) **Information extraction**: First, AC3R extracts information about the crash contributing factors, such as weather and lightning, the vehicles, and the dynamics of the crash. AC3R uses a domain-specific ontology to map concepts in the text to details of the crash scenario.
2) **Trajectory planning**: Second, AC3R creates an abstract representation of the car crash, which includes the geometry of the roads, the initial placement of the vehicles, and their intercepting trajectories.
3) **Simulation generation**: Third, AC3R generates code which implements the dynamics of the car crash and the surrounding virtual environment in the simulator. This also includes runtime checks to determine when the accuracy of the simulations is not satisfactory.
4) **Test generation**: Finally, test cases are derived from the simulations by including test preconditions and test oracles, which check if the *ego-car*, i.e, the self-driving car under test, could avoid the crash.

In the following sections, we discuss each of these steps in more details over a working example. Our working example is a real police report that we took from the National Highway Traffic Safety Administration (NHTSA) database [15] (report number 2005011269283). The following is the relevant extract of the report:

> *The crash occurred on a two-way, two-lane straight, level, bituminous residential street with a speed limit of 25 mph. Conditions at the time of the weeknight crash were cloudy, dark, and dry. V1, a 2001 Kia Sephia, driven by a 28 year-old female, was driving southbound on the road when it left the travel lane and the front of V1 struck the back of a legally parked, unoccupied vehicle on the right side of the road.*

Figure 2 shows the key frames of the simulation which AC3R generated from this report.

A. Information Extraction

AC3R extracts information about weather, lighting, roads, and the vehicles involved in the crash for reconstructing the car crashes. Information extraction leverages standard NLP techniques and works under the assumptions that police reports are grammatically correct and narrate the sequence of events leading to the impact in a chronological order.

1) Report Parsing: During information extraction, AC3R incrementally parses the police reports and accumulates the information about weather, lighting, roads, and vehicles, into a number of data structures which form the abstract car crash.

Information extraction works as follows: First, AC3R computes the *grammatical dependencies* between each pair of words in each statement using the Stanford Natural Language Processing suite [16]. For instance, in our working example AC3R computes the dependencies `nsubj(cloudy, Conditions)`, which identifies "Conditions" as the *nominal subject* of the sentence, and `conj:and(cloudy, dark)`, which connects the two adjectives, and concludes that the weather was cloudy and the lighting was dark. The number of grammatical dependencies in a sentence might be large, but not all the dependencies are essential for reconstructing the car crashes, hence AC3R discards them. For example, dependencies such as punctuations (`punct`) are non-essential, hence discarded.

2) Matching the Domain-Specific Ontology: Next, AC3R tries to match the (stemmed [17]) words mentioned in the remaining grammatical dependencies with instances in a domain-specific ontology, i.e., a hierarchical representation of the concepts, attributes, and relationships [18]. Finding a matching instance in the ontology is the criterion which AC3R uses to distinguish relevant information, which are extracted, from irrelevant information, which are ignored.

AC3R's ontology organizes classes in three levels:

- The *top level* provides the fundamental dichotomy between *environment_property* and car crashes *storyline*.
- The *middle level* identifies structurally complex objects or phenomena which are relevant for reconstructing the car crashes.
- The *bottom level* contains leaf classes, i.e., simple concepts which cannot be further decomposed.

The ontology connects classes at the same level and across levels to structure its knowledge; for example, it connects the middle level class *traffic participant* to its subclass *vehicle* at the same level, and to leaf classes such as *vehicle_action* and *vehicle_type* at the lower level.

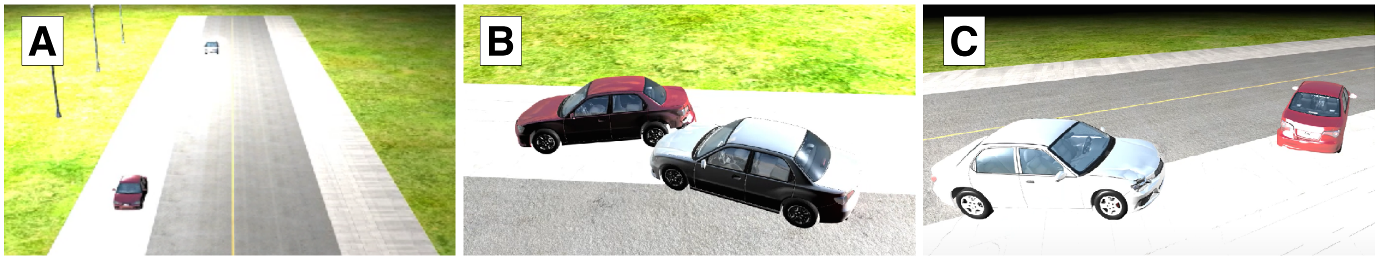

Fig. 2. Key frames of the simulation generated by AC3R from the working example. Frame A shows the virtual cars in their initial position; frame B shows the impact; while, frame C shows the crash aftermath. Video recordings of all the simulations generated by AC3R are available at: (blinded URL)

The ontology also connects classes to named instances, i.e., actual concepts mentioned in the police reports, which characterize the appearance, action, or moving direction of the simulated elements in the reconstructed car crashes. That is, instances bridge the gap between the abstract representation of the car crashes and their implementation by contributing the relevant simulator configuration values. From the working example, the ontology maps the *"Conditions at the time of the crash . . . were . . .* ***dark****"* with an instance of the *lighting* class which is characterized by low value of *lighting_brightness*, and AC3R uses that value for configuring the simulation lighting to achieve the result exemplified in Figure 2.

The basic use of the ontology consists of storing all the attributes values of the matching instances in the corresponding data structures. AC3R selects the data structures to update by looking at which ontology class defines the instance. For example, from `nsubj(cloudy,Conditions)` AC3R matches the instance *cloudy* of the *weather* class; therefore, it updates the data structure which store weather configuration values. Similarly, from `nsubj(dark,Conditions)` AC3R matches the *dark* instance of the *lighting* class and updates the data structure storing lighting informations.

For now, AC3R focuses only on the main crash contributing factors, i.e., lighting, weather, roads, and cars movement; hence, it neglects other details, such as traffic signs and signals, that might be important to accurately reconstruct the environment in which crashes happened. Additionally, we keep AC3R's ontology illustrative and simple enough to cover the police reports considered in our evaluation; hence, AC3R's ontology does not store all the possible variations of weather conditions. Extending AC3R to include those missing elements is conceptually simple, and is part of our ongoing work to improve the accuracy of the recreated car crashes.

3) Extracting road properties: While categorical information about roads, like the material used to pave them, is extracted as described above, a different approach has to be adopted to extract information about quantitative properties of roads, such as their speed limit or their slope. For these properties, AC3R builds dependency chains linking the quantitative properties, their measurement units, and their actual values. In our working example, AC3R finds that `nsubj(mph,speed limit)` and `nummod(mph,25)` are related, hence it stores the value 25mph into the abstract car crash.

Elaborating descriptions of crashes which happened at intersections is more challenging because statements referring to single roads are mixed with statements describing multiple roads at once, and because descriptions refer to roads *indirectly* by using their cardinal direction (e.g., the "East/West" roadway). AC3R handles these cases by looking at the cardinality of nouns and updating all the road information at once when the plural form is used, and by matching roads by the value of their direction attribute.

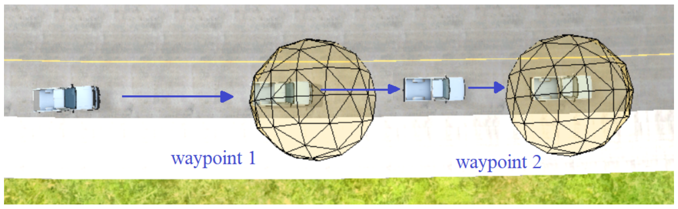

Fig. 3. Trajectory planning using waypoints (represented by BeamNG.drive as semi-transparent spheres). A sequence of waypoints defines a set of constraints on the trajectory of the cars which must pass through each one of them in the specified order.

4) Extracting vehicle properties: Next, AC3R extracts the information about the vehicles involved in the crash and the driving actions which lead to it. Vehicle descriptions are elaborated using the basic approach of matching instances in the ontology and updating the corresponding data structures, but driving actions require a different approach. AC3R identifies driving actions as those verbs which 1) match instances of the class *vehicle_action*, and 2) whose subjects or objects are instances of the *vehicle* class. AC3R distinguishes regular driving actions, like "travel" and "drive", from verbs, like "hit" or "struck", which describe an impact. For each type of driving action, AC3R extracts the most appropriate information: the speed value and the direction of movement for regular driving actions; and, the references to the striker and victim cars as well as the components damaged in the impact otherwise.

In our working example, AC3R matches "struck" with an instance of an impact driving action, and infers from `nsubj(struck, front)` and `nmod:of(front, vehicle1)` that vehicle1 is the striker and the impact damaged its front; similarly, AC3R infers that vehicle2 is the victim and the impact damaged its back.

B. Trajectory Planning

After extracting the information about the driving actions, AC3R computes the trajectory that the virtual cars must follow in the simulation to recreate the impact; coincidentally, AC3R defines the geometry of the road which must simply comply with the direction of car movement.

Trajectories are implemented as sequences of *waypoints*, i.e., positions in the tridimensional space which the virtual cars must visit in order. Figure 3 illustrates this concept by showing a virtual car going through two waypoints. By placing one of the waypoints at the impact location and sharing it with the vehicles involved in the crash, AC3R ensures that they crash into each other during the simulation.

Trajectory planning starts by placing the impact point on the origin of the reference coordinates space; then, it continues *backwards* by placing a new waypoint for each of the driving actions registered for each virtual car. AC3R keeps adding waypoints until all the driving actions are included; the last waypoint added, which corresponds to the first driving action mentioned in the police report, automatically identifies the initial positioning of virtual cars.

Initially, the impact point is placed on the center point of a straight road segment; AC3R might adjust the position of the impact point during the trajectory planning because the crash involves a parked car, in this case AC3R translates the impact point to the side of the road, or because the road is *curvy*, in this case AC3R updates the road geometry and moves the impact point at the new location.

AC3R creates first the trajectory of the striker car. For each driving action associated to the striker car, AC3R uses basic trigonometry and a simplified kinematics model to compute a segment of the trajectory. It does so assuming that the striker car travels at constant acceleration to reach the expected speed for the segment, starting from the previously placed waypoint and moving in the specified direction. AC3R ensures that the striker has just enough room to reach the specified speed values before arriving at the next waypoint; this ensures that the simulation matches the police report but also limits the overall duration of its execution. The victim car might be parked or it might move. If the victim car is parked, AC3R places the victim car directly on top of the impact point, ensuring this way the striker will hit the victim. If the victim car moves, first AC3R computes the victim's trajectory in isolation; then, it adjusts the striker trajectory to hit the victim by computing the an intercept trajectory.

C. Simulation Generation

AC3R relies on BeamNG.drive [19], an extensible video game engine which features accurate soft-body physics and realistic 3D textures of roads, vehicles, weather and lighting conditions, for simulating the car crashes.

BeamNG.drive is specialized in simulating traffic accidents: It offers comprehensive mechanisms for controlling the virtual cars during the simulation, but it also exposes the runtime data about vehicle positions, speed, forces, and damaged components, which AC3R requires to validate its crash reconstructions. Further, BeamNG.drive enables external software to monitor and control the virtual cars, which is fundamental for testing self-driving car software using simulations [20], as in our evaluation.

To simulate a car crash using BeamNG.drive, AC3R creates a number of files to configure aspects of the simulations, like the general information about the driving scenario and its lighting, the terrain map, the road geometry and texture, and the cars in their initial positioning. AC3R uses the configuration values about weather, lighting, road dimensions, and so on, defined in the ontology and mapped to the abstract car crash during the information extraction step. Additionally, it uses the geometrical data computed while planning the trajectories of the virtual cars to setup the roads and waypoints.

AC3R generates code using a customized, template-based approach: For each of the relevant elements in the abstract car crash, the generator produces a snippet of code which instantiates the corresponding object in the simulator. AC3R also generates a script, written in the LUA scripting language [21], which contains the dynamic elements of the simulations, including the code which triggers the movement of the cars and the runtime checks to assess simulation accuracy.

The simulation code can be executed in order to replay the car crashes generated from the police reports. This enables testers to manually validate if the generated simulations match their expectation. Additionally, while AC3R replays the car crashes, it collects data about the damaged components. After the simulation, AC3R verifies the accuracy of the simulation by checking if an impact was registered, and if the damaged components described in the police reports match those reported by the simulator.

D. Test Generation

The simulations generated by AC3R contain all the necessary information to recreate the car crashes described in the police reports; however, these simulated car crashes do not qualify *per-se* as test cases because they lack proper test oracles and do not allow any exogenous interference. That is, the generated simulations cannot automatically check if the behavior of the ego-car is acceptable, and the ego-car cannot freely interact with the simulated environment. Therefore, we elaborate AC3R's output to derive proper test cases. Such test cases need to meet some basic requirements; in particular, they shall (i) be relevant for the driving task at hand; (ii) not alter the semantics of the original car crashes; (iii) fit the ego-car's specifications; and, (iv) be robust in the face of the stochastic nature of physics simulations. In order to meet these quirements, AC3R applies the following considerations when generating test cases.

1) Choosing which virtual car the ego-car drives: Car crashes usually involve more than one vehicle, hence also AC3R may simulate more than one virtual car. In principle, the ego-car can control all the virtual cars; however, the choice of which virtual car is actually controlled by the ego-car has an impact on the relevance, semantics, and validity of the resulting test cases. For example, if the driving task is to *"avoid crashing into a car legally parked on the side of the road,"* then configuring the ego-car to control the parked car, i.e., the victim, is not as relevant as letting it drive the striker car which originally caused the crash. Similarly, if the ego-car drives the parked car away from its original position before the striker car reaches it, then the striker car, whose motion is pre-configured in the simulation, ends up driving out of the road. This effectively changes the original semantics of the car crash from *"impacting a legally parked car on the side of the road"* into *"going off the road."*

2) Making oracles robust: Tests based on simulations produce continuous data, hence defining equality conditions to decide when a test passes or fails might result in brittle tests. We avoid this by defining three robust oracles that check conditions on waypoints, damaged components, and elapsed time. For each test, we define a driving task which consists of reaching a goal waypoint, or its vicinity, placed behind the impact location computed by AC3R, within a given timeout and without crashing. We define a permissive timeout which depends on the speed limit on the road segment and the estimated travel distance to reach the goal waypoint. For crashes, we strictly check that none of ego-car's components report any damage to ensure that tests fail if the ego-car crashes into the designated victim or any other car, but do not fail if others car crash into each other.

3) Defining test preconditions: AC3R recreates car crashes by pre-programming the movement of virtual cars using waypoints and distance-based triggers. Distance-based triggers decide when virtual cars start to move based on their location, and are evaluated at every frame. Since the rendering engine used for the simulation does not guarantee a constant frame rate, distance-based triggers are not always evaluated at the same time. Further, vision-based self-driving car software assumes constant frame-rates and might compute inaccurate driving actions under variable frame-rates. These factors cause nondeterminism in the tests which affects their quality and might result in *flakyness* [22]. To avoid this, we define test preconditions as tolerance intervals around the reference triggering distance. Test preconditions invalidate the test executions when the difference between the reference and actual distance values exceeds a threshold. This reduces the occurrence of tests which fail because of *unavoidable* crashes due to belayed cars movement. For instance, in one of the cases we investigated the victim leaves its parking position while the striker is approaching; in this case, if the victim moves too late, the striker is too close to it and has not chance to avoid the impact no matter which algorithm drives it.

III. Evaluation

In order to empirically evaluate AC3R, we aim to study how closely the simulations generated by AC3R match the descriptions from the police reports (RQ1, RQ2), how efficient the technique is (RQ3), and whether the police reports are suitable as test scenarios (RQ4).

In line with previous work on reconstructing car crashes for validating ADAS systems [14], we first evaluate the accuracy of our simulations "in-the-small," and formulate the first research question:

TABLE I
Details of the NHTSA police reports for the evaluation of AC3R.

#	Case ID	Crash type	Summary
1	2005008586062	Frontal Impact	Driver hit parked car in day-light
2	2005011269283	Frontal Impact	Driver hit parked car at night
3	2005008586061	Frontal Impact	Driver hit parked among many
4	–	Sideswipe	Driver hit car exiting a parking spot
5	2005002585724	Straight Path	Driver ran a stop at 2-way junction
6	2005004495041	Turn Into Paths	Driver did not yield at T-junction

We obtained Case #4 by extending the description from Case #3 to include an additional element of complexity, hence increase the coverage of crash types.

RQ1 *Do the simulations generated by* AC3R *culminate in the expected impact?*

Answering RQ1 lets us understand if the virtual cars get damaged in a way which is compatible with the crash descriptions. This, in turns, helps testers in checking if the reactive components of self-drivings cars, like emergency brakes, avoid the impacts or at least reduce their criticality.

In line with previous work on visualizing car accidents [23] and reconstructing critical road segments [24], we also evaluate the accuracy of our simulations "in-the-large":

RQ2 *Do the simulations generated by* AC3R *accurately reconstruct the environment surrounding the crashes as well as their overall development?*

Answering RQ2 lets us understand if the simulations contain all the necessary visual and behavioral elements for reproducing the car crashes realistically. This, in turns, helps testers in checking if the mid-term planning components of self-driving cars, like adaptive cruise controllers and lane keeping algorithms, would have prevented the car crashes in the first place.

In order to assess the practical applicability of AC3R, we study how efficient AC3R is at analyzing the police reports and generating the corresponding simulations to be used within regular development activities:

RQ3 *How efficient is* AC3R *in generating accurate crash simulations from police reports?*

Finally, we would like to understand how effective the tests derived from the simulations are:

RQ4 *Do virtual tests derived from real car crashes find shortcoming and problems in self-driving cars?*

A. Experimental Setting

For answering the research questions, we sampled the National Motor Vehicle Crash Causation Survey database of the National Highway Traffic Safety Administration (NHTSA) [15]. The crash reports in the NHTSA database are multi-media semi-structured XML documents which store information of crash contributing factors in specific tags and describe the development of crashes by means of short texts. Notably, NHTSA reports follow the Model Minimum Uniform

Crash Criteria (MMUCC) guidelines [25] which state that crash events must be narrated in chronological order. That is, NHTSA reports fulfill AC3R's main working assumption.

To answer RQ1–3 we selected five NHTSA crash reports, and used AC3R to generate the corresponding simulations. We selected these five police reports because they describe crash dynamics and environments of various complexity, and cover various types of car crash, i.e., "Frontal Impact", "Straight Path", and "Turn Into Paths." For example, cases #1 – #3 describe striker cars frontally impacting parked cars; Case #1 develops in an almost straight road during day-light; Case #2 develops at night; and, Case #3 develops in a narrow road with cars parked on both sides. Cases #5 and #6, instead, involve moving cars impacting at intersections and resulting in different lateral collisions. These five reports cover three out of the nine car crash types reported in the NHTSA database. In an effort to cover more crash types without increasing the complexity of the environmental setup, we manually extended the description of Case #3 by including statements which describe parked car leaving its parking position and obtained Case #4. Compared to the other reports, Case #4 covers one additional car crash type (i.e., "Sideswipe"). In summary, this evaluation considered the six police reports listed in Table I, which in total cover four out of the nine crash types mentioned in the NHTSA database.

To answer RQ4 we derived test cases from these six simulations, and use those to test *DeepDriving*, a state-of-art self-driving car [26]. Specifically, in this evaluation we tested the implementation of DeepDriving based on TensorFlow [27] provided by Netzeband [28], which was trained using images collected from 14 hours of game play of the racing game called SpeedDreams [29] and achieved smaller error rates compared to the original implementation by Chen et al. [26].[1]

For the evaluation, we executed AC3R and BeamNG.drive on a commodity *gaming* PC equipped with an AMD Ryzen 7 1700X 8-Core CPU working at 3.4 GHz, 64 GB of Memory, and an NVIDIA Geforce GTX 1080 GPU.

B. Threats to Validity

a) Internal validity: To ensure that our integration of DeepDriving into BeamNG.drive is correct, we used the author's guidelines, manually tested the integration, and validated DeepDriving by having it successfully drive along randomly generated roads. While AC3R does not currently recreate all elements (e.g., traffic signs), all required elements specified by DeepDriving are included in the reconstruction. To validate that BeamNG.drive correctly reports the damage of components, we visually verified the video recordings from the test executions and ensured that the passing tests did not involve any car crashes. To avoid bias in the user survey, we anonymized it and assigned tasks to participants randomly.

b) External validity: We manually selected a small number of police report from the NHTSA database, because the elaborate evaluation including the user study does not easily scale. However, we ensured that the selected reports cover different crash types, lighting conditions, road geometries, amount of cars, and car movements. Our user study, like any empirical investigation, involved a limited amount of participants, so results might not generalize. We tried to diversify respondents by involving participants of different ages, from industry and academia, and with decent technical experience. We only used a single self-driving car software for evaluation, because most self-driving car software is not publicly available. While we therefore cannot say if effectiveness results generalize, our evaluation nevertheless demonstrates that AC3R is able to generate relevant test scenarios.

[1]The result of this comparison are available at:
https://bitbucket.org/Netzeband/deepdriving/wiki/DeepDrivingEvaluate

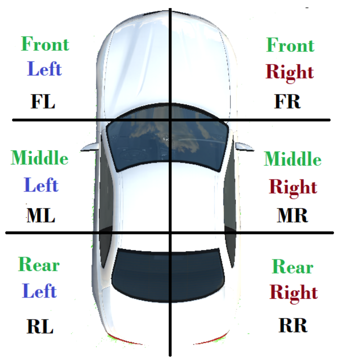

Fig. 4. Labelling schema for the damaged car components reported by BeamNG.drive.

C. RQ1: Accuracy of Simulated Impacts

RQ1 investigates how closely the impacts reconstructed by AC3R match their descriptions from the police reports. We evaluate this by comparing the descriptions of the damaged components from the police reports, e.g., "front-left", "back", with the damaged components as reported by BeamNG.drive according to the labeling schema depicted in Figure 4.

We qualify the accuracy of the impact for each car by means of two binary variables, P and S. P accounts for the damaged parts of the car (i.e., "front", "middle", and "rear"), while S accounts for the car side (i.e., "left" or "right"). Note that we match labels considering the possibility of the police reports being underspecified. For example, if the simulator reports damage in the "front-left" and the police report mentions "front-right", then we have a partial match since only P matches; however, if the police report mentions only "front", then a match of P results in a total match, regardless of the label for S. Having computed the values for P and S, we label simulations as:

- *Total match* (TM), if both P and S match for all cars;
- *No match* (NM), if P or S do not match for all cars,
- *Partial match* (PM), in all the other cases.

With these accuracy labels, we qualify the results achieved by AC3R by means of precision and recall. *Precision* measures the ability of AC3R to construct crashes which are at least partially matching. We compute precision as the ratio of partial and total matching cases over all the cases. Values of precision close to one indicate that AC3R created simulations

TABLE II
RESULTS FROM THE IMPACT ANALYSIS.

#	Striker Car		Victim Car		Label
	P	S	P	S	
1	✓	✓	✓	✓	TM
2	✓	✓	✓	✓	TM
3	✓	✓	✓	✓	TM
4	×	×	✓	✓	PM
5	✓	✓	✓	✓	TM
6	✓	✓	✓	✓	TM

In the table, '✓' indicates conditions which match between police reports and simulations; '× ' indicates conditions which do not match. 'TM' indicates total matches, 'PM' partial matches, and 'NM' no matches.

which match the police reports for almost all the involved cars. *Recall*, instead, quantifies the fraction of matching cases with most desirable result. We compute recall as the ratio of total matching cases over all the matching cases. Values of recall close to one indicates that AC3R created simulations that literally resemble the descriptions from police reports.

Table II reports the accuracy results of the simulations generated from the six police reports considered in this evaluation. The table reports for each case its identifier (#), the values of P and S for the striker and victim cars, and the resulting accuracy label. A check-mark (✓) indicates a match between description and simulation, while the x-mark (×) identifies a condition which did not match.

From the data reported in Table II, we compute the following values for precision and recall: Precision $= 1.0$, and Recall $= 0.83$, which indicate that in all the simulations generated by AC3R at least one of the two virtual car reported the expected damage, and that in the majority of the simulations all the virtual cars impact as described. Only in Case $\#4$ we could not match the striker car damage. The reason for this is a limitation in the BeamNG.drive collision detection which did not report any damage for the striker in this specific case of car swiping. We visually verified that in Case $\#4$ the striker car impacted on the correct side; however, since the simulator reported no damage, we conservatively opted for considering all the labels of the striker car as not matched. In summary, we conclude that:

AC3R *can accurately reconstruct the expected impact between virtual cars*

D. RQ2: Accuracy of the Overall Simulations

RQ2 investigates how closely the environment reconstruction and the development of the whole car crashes (i.e., not just the way virtual cars impact) match the police reports.

Since the police reports narrate the development of car crashes using natural language which must be interpreted, we answer RQ2 empirically by means of a user study. Specifically, we asked responders to 1) read the textual descriptions of the crash reports; 2) watch the videos that we recorded from the corresponding simulations; and, 3) express their agreement with a series of statements about the reconstructed car crashes. During the study, responders were free to read the descriptions and watch the videos as many times as needed.

TABLE III
STATEMENTS FROM THE USER STUDY.

ID	Statement
1	Overall, the accident develops as expected.
2	The vehicles are initially positioned as described.
3	The vehicles move as described.
4	The crash happens at the expected location.
5	The damaged side (Left Rear, Front, Right Front, etc.) of each vehicle matches with the crash report.
6	The simulated crash happens naturally.
7	The environment reconstruction (number of roads, roads type and direction, weather, etc.) matches the description.

In our study, responders evaluate each statement using a 5-point Likert scale, ranging from strong disagreement to strong agreement.

Table III lists the seven statements we used for evaluating the main aspects of the simulations which include the general development of the simulation (#1), the setup and movement of virtual cars (#2 and #3), and different aspects about the crash (#4 – #6) and the environment (#7) reconstructions. For each statement, we used a 5-point Likert scale [30], ranging from *"Strongly Disagree"* to *"Strongly Agree,"* to capture responders degree of agreement. We chose a 5-point Likert scale to give responders enough expressive power while keeping a clear separation between the response categories. We also gave responders the possibility to provide additional feedback in the form of short textual answers. In addition to the five categories of the Likert scale, we included the *"Don't know/Don't want to answer"* category. This increased the robustness of our results towards the occurrence of random responses because it gives responders a default category to select when they could not form an opinion about the statements.

We asked responders to evaluate only three (out of six) cases to balance the trade-odd between the effort spent by the responders for completing the survey and the benefit of collecting more data for our analysis. A preliminary interview with a subset of the responders revealed that on average it took them about 20 minutes to evaluate three cases.

Responders participated to our survey anonymously and voluntarily, i.e., we did not use any monetary incentive or prize to convince them to complete the survey. However, the survey includes a few optional biographical questions to capture responders' age, technical background, and driving capabilities. Analyzing the demographics of the responders let us contextualize the results of our analysis and assess the dependability of the responders.

We ran our study using a standard online survey service and invited a total of 80 persons via email. We invited a fairly heterogenous set of participants, including university students at all levels, professional researchers, and professors at different universities. We also extended the invitation to BeamNG developers and the authors' personal contacts.

A total of 34 out 80 responders (response rate= 42.5%) completed the survey. Responders are aged between 18 to over 40, and many of them (76%) claimed driving experience. Given these figures, we assume that responders can objectively evaluate the various aspects of the generated simulations.

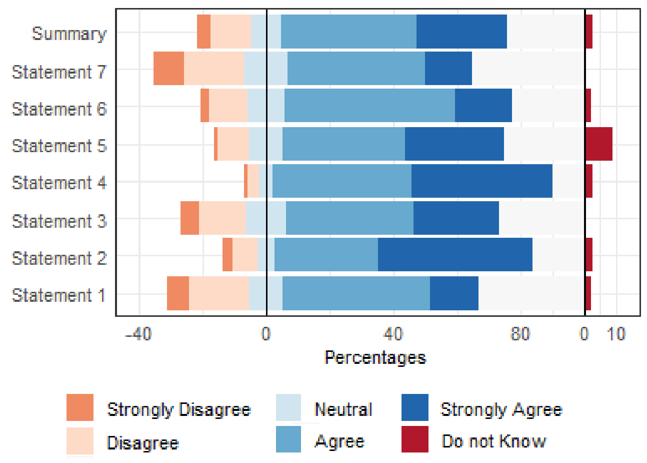

Fig. 5. Aggregated responses per statement to evaluate the accuracy of reconstructed car crashes. Table III reports the extended formulation of the statements.

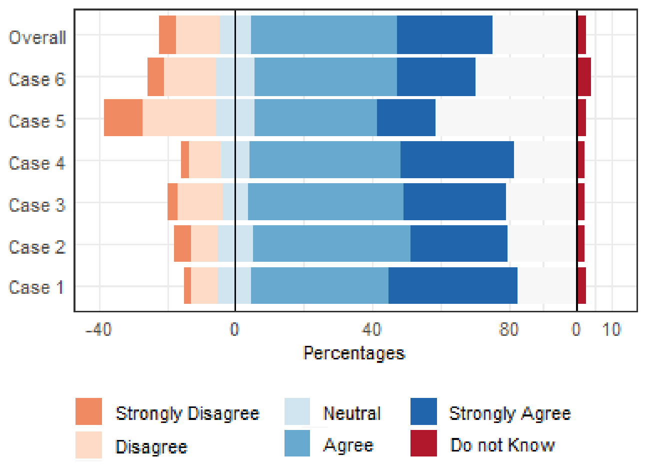

Fig. 6. Aggregated responses per simulation to evaluate the generality of the approach. Table I reports the details of the car crashes.

The 34 responders provided a grand total of 714 responses and 240 optional comments; however, for the sake of brevity, we report in this section only a summary of our main findings. The interested readers can find the entire dataset on AC3R's web page. We aggregate the responses by computing in which proportion each of the categories was selected, and report two complementary aggregations of the data as diverging stacked bar charts in Figure 5 and Figure 6. Diverging stacked bar charts report data as horizontal bars which stack positive responses to the right of a vertical baseline and negative responses to its the left, and are the standard way to visualize rating scales like Likert data [31]. We placed the *"Don't know/Don't want to answer"* responses in their own column since these responses do not belongs to any category defined by the 5-point Likert scale.

Specifically, Figure 5 visualizes the responses aggregated *per statement* to assess how accurately AC3R handles the different aspects of crashes reconstruction; conversely, Figure 6 visualizes the responses aggregated *per simulation* to draw conclusions about the generality of our approach.

From the results reported in Figure 5 and Figure 6 we can observe that responders considered the reconstructed scenarios to be a good match with the descriptions across the seven statements, and AC3R can accurately reconstruct many different cases (the median value is *Agree* in both aggregations). From Figure 5, we observe that responders mostly agreed that AC3R accurately recreated the initial and final states of simulations (i.e, Statements #2, #4 and #5), and that simulated crashes developed *naturally*, i.e., responders did not feel simulations as artificial (Statement #6).

Responders still agree, although less strongly, with the statements about the environment (Statement #7), the overall development of the crash (Statement #1), and the movement of virtual cars (Statement #3). From Figure 6, we observe that responders largely agreed that AC3R accurately reconstructed cases #1 – #4, while it was less accurate for cases #5 and #6. Analysis of the optional comments provided by the responders let us identify the reasons of this. Regarding environment reconstructions, responders explained that the main sources of inaccuracy were the lack of traffic elements and the improper setup of weather. This was particular evident for cases #5 and #6 which describe crashes happened at traffic junctions, and results from the incompleteness of our prototypical implementation.

Regarding the overall development of the crash and the movement of the virtual cars, responders identified the main source of inaccuracy in the way AC3R simulated the crash aftermath. That is, virtual cars behave as described up to the first impact, but then did not rotate, keep moving, or push each other as expected. This result is expected and follows from our main design choice to recreate only the *first* impact mentioned in the police reports. Being able to accurately reproduce the crash aftermath, which is part of our future plans, increases the perceived accuracy of the simulations; however, not doing it is not necessarily detrimental for testing self-driving cars which must avoid the first impact in any case.

By analyzing the distribution of *"Don't know/Don't want to answer"* we observe that our results are robust since the majority of the responders (more than 95% on average) formed an opinion about all the statements in all the cases, and even in the worst case (Statement #5 in Figure 5) only 8.8% declared *"Don't know/Don't want to answer"*. The analysis of the optional comments associated with Statement #5 revealed that in some cases responders selected *"Don't know/Don't want to answer"* because police reports lacked the details about impact, so responders could not agree nor disagree about the accuracy of the impact reconstruction. Despite this, responders largely agreed that AC3R accurately reconstructed the impact, which also strengthens the results we obtained for RQ1. In summary, we conclude that:

AC3R *can accurately reconstruct diverse car crashes and their surrounding environment.*

E. RQ3: Efficiency of AC3R

RQ3 investigates the practical applicability of AC3R by measuring its efficiently in generating the simulations. There-

TABLE IV
SUMMARY OF THE CAR CRASHES UNDER STUDY AND THE OUTCOME OF THE TEST CASES DERIVED FROM THEM.

#	Driving Task	Distinguishing Features	Test Outcome	Additional Observations
1	Keep lane	Pass	Curved road, day light	Parked vehicle identified, no oscillations in steering.
2	Keep lane	Pass	Straight road, night	Road type misinterpreted
3	Keep lane	Pass	Parked cars, evening	Oscillatory steering
4	Follow car	Fail	Car leaves parking spot	Collision avoided but DeepDriving stuck of the road
5	Pass intersection	Skip	Car runs on stop, day light	Requires side view, DeepDriving can not handle this case
6	Pass intersection	Skip	T-Junction, Car runs on stop	Requires side view DeepDriving can not handle this case

The table reports for each of the test cases derived from the six simulations generated by AC3R, the unique identifier, the main driving task to handle, and the test outcome. We complement this with the list of distinguishing features for each test case, and additional observations collected during tests execution.

fore, we answer the question by measuring AC3R's execution time to process the six police reports listed in Table I.

Across five repetitions, AC3R took 27.2 seconds on average for reconstructing each simulation, and most of the time was spent during information extraction. The descriptions of all the cases are rather compact. They range from 101 words in Case #1 up to 203 words in Case #5, with median value of 118 words across the six descriptions; however, the time required to extract information out of them is binomially distributed: processing cases #1, #3, #4, and #6 took 16 seconds on average, while processing cases #2 and #5 took about three times longer. The reason for this remarkable different is that processing cases #2 and #5 requires coreference resolution, an extremely expensive NLP task.

According to an informal interview with the experts at BeamNG, manually generating each of these six simulations would easily take up to two hours; hence, we conclude that:

AC3R *is efficient and can be used in practice for generating crash simulations from police reports.*

For reference, we also computed the execution time for the simulations which took on average 22 seconds across the six cases considered here. Simulation time and generation time are only weakly connected. The former depends mostly on the amount of driving actions mentioned in the reports, while the latter depends mostly on how the officers wrote them.

F. RQ4: Effectiveness of AC3R *Test Cases*

RQ4 investigates the ability of the test cases derived from AC3R simulations to expose problems in state-of-art self-driving cars. Therefore, we derived test cases from the six simulations considered in the previous evaluation, and used them to test DeepDriving by integrating it with BeamNG.drive.

DeepDriving is a vision-based self-driving car which follows the *direct perception* paradigm. According to Chen et al. [26], direct perception is one of the three major paradigms to design self-driving car software. According to it, self-driving cars use Convolutional Neural Network (CNN) [32] to estimate various driving affordance indicators, such as the distance to the vehicles in front, or the position of the ego-car on the road, and a standard controller to compute the driving actions from them. For example, DeepDriving utilizes a rule-based controller to keep the ego-car in the middle of lane, slow down while turning, and stop before impacting into obstacles. Notably, DeepDriving recognizes cars and road markings, but ignores other elements such as traffic signs and signals; hence, the crashes reconstructed by AC3R match DeepDriving's requirements.

The tests we derived from the simulations always configure DeepDriving to drive the *striker* cars. This preserves the original semantics of the driving tasks for test cases #1 – #4 which test the abilities of the ego-car to keep the lane and avoid frontal collisions. Test cases #1 – #4 require only the frontal view of the road, hence they fit with DeepDriving's specifications. In contrast, test cases #5 and #6, which test if the ego-car can avoid lateral collisions, require side view images; hence, they unfortunately do not fit with DeepDriving's specifications, and we can not use them to test it.

Table IV summarizes the results of this evaluation by reporting, for each of the test cases, its identifier (#), the main driving task faced by the ego-car, and the test outcome. The table also lists the main distinguishing features of the test cases and additional observations that we collected during their execution.

From the results in Table IV, we observe that DeepDriving successfully completed three test cases, and failed one. Test case #4 failed because DeepDriving did not successfully complete the driving task specified by the test, that is, it did not reach the goal waypoint in the allotted time.

During test case #4, DeepDriving performed an evasive maneuver to avoid the victim car which suddenly exits its parking space; however, after avoiding the victim, DeepDriving did not move anymore, even though the road ahead was free and safe to drive. Since all the indicators estimated by DeepDriving's CNN during the execution of test case #4 were compatible with the dynamics of the crash as described in the police reports, we identified the root cause of this anomalous behavior in a design flaw of the controller: In short, the controller computes a slow-down factor to reduce its speed using the intensity of the past five steering values; this is a sensible heuristic to drive on turns since it reduces the ego-car speed proportionally to the sharpness of turns. However, the controller implicitly assumes that DeepDriving steers only while turning while its current speed is always larger than the slow-down factor. Performing the evasive maneuver violated both assumptions and forced the controller into a state of *negative feedback*: the controller steers for moving the car to the center of the lane from its current position, but it also brakes because of the steering command, hence it remains in the current position, and the cycle repeats. Finding this bug is not trivial: A test which can expose this problem must cause DeepDriving to issue consecutive steering commands such that the sum of their values is larger than its current speed *while* its position on the road is off the middle of the current lane enough to ensure the controller keeps steering.

DeepDriving passed the other tests, but the test executions revealed further (not safety critical) problems, described in Table IV, col. *Additional Observations*. For example, in test case #3, DeepDriving's steering resulted in strong oscillations. In this test case, DeepDriving drove on a very narrow road with cars parked on both sides; as DeepDriving approached the parked cars, it tried to make room for them by steering in the opposite direction; however, since these were cars parked on both sides of the road, DeepDriving ended up repeatedly steering left and right. While AC3R does not currently include a test oracle for this behavior in the generated tests, this behavior may cause passengers nausea and motion sickness [33], and shall be avoided [34]. In the light of these results, we positively answer RQ4:

The test cases we derived from the real car crashes indeed expose problems in a state-of-art self-driving car.

IV. Related Work

CarSim [35], similarly to AC3R, reconstructs car accidents from short textual descriptions. In contrast to AC3R, CarSim works with inaccurate descriptions from newspapers and aims to support humans in understanding them by means of visualization. That is, CarSim is not subjected to the same strict requirements on simulations fidelity as AC3R is, hence CarSim achieves lower accuracy [23]. Erbsmehl [14], instead, proposes to recreate car crashes by replaying the sensory data collected during the crashes. This limits applicability, because it is based on naturalistic field operational data which are not generally available. Further, Erbsmehl's simulations cover only up to three seconds before the impact, hence they are not suitable to test self-driving car software at system level as done by AC3R, which generates instead simulations covering a longer timeframe (i.e., 22 seconds on average).

Existing approaches to systematically generate virtual tests adopt various techniques: Sippl et al. [36] identify relevant test cases by mining large amounts of simulation data; Mullis et al. [37] find relevant test cases by identifying the performance boundaries of the system under test; Abdessalem et al. [9], [11] search for test cases which maximize an application-specific fitness function; and, Tuncali et al. [38] search for test cases which falsify safety properties of the cars. While these approaches aim to generate large amounts of test cases in the hope of including relevant, critical ones, AC3R directly generates relevant test cases from real car crashes.

In traditional software testing, some approaches adopt a similar philosophy for generating relevant tests from crash data collected during failing executions [39]–[42]. These approaches are conceptually similar to the work proposed by Erbsmehl, with the fundamental difference that in-depth data about software crashes are available in large quantity. Fazzini et al. [43], instead, generate tests from bug reports by combining program analysis and NLP. This approach follows the same insight underlying AC3R: bug reports, like police reports, identify cases that should have not happened, and must be tested for to avoid their future occurrence.

AC3R utilizes Natural Language Processing and a domain-specific Ontology to generate test cases. In the automotive domain, Wang et al. [44] generate system tests using NLP on use case specifications, but their application-specific test oracles at the component level [45] might not be enough to check if car crashes are avoided. Armand et al. [46], [47], use domain-specific ontologies, but apply these for interpreting and contextualizing sensory data collected by self-driving cars.

We applied AC3R to test a state-of-art vision-based self-driving system. Current approaches to test vision-based self-driving systems instead exploit the fact that those systems are based on convolutional neural networks, hence they use adversarial techniques for generating images which cause the convolutional neural networks to make poor predictions [48], [49]. While these adversarial techniques require extensive knowledge about the internals of the CNNs to generate single images, AC3R is completely black-box and generates entire simulations, hence is more general.

V. Conclusions and Future Work

In this paper we presented AC3R, a novel automatic text-to-scene conversion approach for generating computer simulations, which reproduce real car crashes, and test cases from police reports. The virtual tests generated from car crashes can test self-driving car software under realistic and critical conditions. Our extensive evaluation showed that AC3R generated efficiently accurate simulations of diverse critical scenarios, and our virtual tests exposed unexpected behavior and a critical bug in thoroughly evaluated self-driving car software.

AC3R takes a fresh approach to generate simulation-based tests for self-driving cars, however, the approach is still in its infancy and further research is needed. Our ongoing work include extending AC3R's knowledge base to address the lack of details in the environment reconstruction and in the simulation of crashes aftermath. As identified by our extensive user study this will improve the accuracy of our simulations. Additionally, we plan to extend our evaluation to larger types of car crashes involving more than two cars and other traffic participants, i.e., pedestrians and cyclists.

Future extensions also include the following:

- **Sophisticated traffic and drivers models** can replace the basic kinetics model used for trajectory planning, increasing the precision of the reconstruction dynamics [50].
- **Evolutionary testing** can be used to generate more test cases from the same police reports. All these tests will implement the same high level driving scenario under a number of different execution conditions [51].
- **Official traffic regulations** can be formalized into automatically verifiable formulae [52], which will provide more and better test oracles.

References

[1] M. Bertoncello and D. Wee, "Ten ways autonomous driving could redefine the automotive world," *McKinsey & Company, Automotive & Assembly*, June 2015.

[2] Department of Motor Vehicles, "Report of traffic accident involving an autonomous vehicle." http://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/autonomousveh_ol316\%2B, 2006.

[3] F. Lambert, "Understanding the fatal tesla accident on autopilot and the nhtsa probe," *Electrek*, 2016.

[4] The Guardian, "Self-driving uber kills arizona woman in first fatal crash involving pedestrian." https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe, February 2018.

[5] ArsTechnica, "NTSB: Uber's sensors worked; its software utterly failed in fatal crash." https://arstechnica.com/cars/2018/05/emergency-brakes-were-disabled-by-ubers-self-driving-software-ntsb-says/, May 2018.

[6] 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.

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

[8] 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.

[9] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pp. 63–74, 2016.

[10] 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 preprint arXiv:1807.00412*, 2018.

[11] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proceedings of the International Conference on Software Engineering*, ICSE 2018, pp. 1016–1026, 2018.

[12] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," in *NASA Formal Methods Symposium*, pp. 357–372, Springer, 2017.

[13] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Sim-ATAV: Simulation-based adversarial testing framework for autonomous vehicles," in *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, HSCC 2018, pp. 283–284, 2018.

[14] C. Erbsmehl, "Simulation of real crashes as a method for estimating the potential benefits of advanced safety technologies," in *Proceedings of the international technical conference on the enhanced safety of vehicles*, no. 09-0162 in ESV 2009, pp. 1–10, 2009.

[15] National Highway Traffic Safety Administration (NHTSA), "National motor vehicle crash causation survey." https://crashviewer.nhtsa.dot.gov/LegacyNMVCCS/Search.

[16] M.-C. de Marneffe and C. D. Manning, "The stanford typed dependencies representation," in *Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, CrossParser 2008, pp. 1–8, Association for Computational Linguistics, 2008.

[17] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[18] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.

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

[20] W. Huang, K. Wang, Y. Lv, and F. Zhu, "Autonomous vehicles testing methods review," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, ITSC 2016, pp. 163–168, Nov. 2016.

[21] R. Ierusalimschy, *Programming in Lua, Fourth Edition*. Lua.Org, 2016.

[22] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pp. 643–653, 2014.

[23] R. Johansson, A. Berglund, M. Danielsson, and P. Nugues, "Automatic text-to-scene conversion in the traffic accident domain," in *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI 2005, pp. 1073–1078, 2005.

[24] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *IEEE Intelligent Vehicles Symposium*, IV 2017, pp. 719–726, June 2017.

[25] National Highway Traffic Safety Administration (NHTSA), "MMUCC model minimum uniform crash criteria second edition 2003." https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/809577, 2003.

[26] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," in *Proceedings of the International Conference on Computer Vision*, ICCV 2015, pp. 2722–2730, 2015.

[27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, OSDI 2016, pp. 265–283, USENIX Association, 2016.

[28] A. Netzeband, "Deepdriving for tensorflow V1.0." https://bitbucket.org/Netzeband/deepdriving/src, 11 2017.

[29] A. Netzeband, "Deep-SpeedDreams." https://bitbucket.org/Netzeband/deep-speeddreams, 11 2017.

[30] R. Likert, "A technique for the measurement of attitudes.," *Archives of psychology*, 1932.

[31] R. Heiberger and N. Robbins, "Design of diverging stacked bar charts for likert scales and other applications," *Journal of Statistical Software, Articles*, vol. 57, no. 5, pp. 1–32, 2014.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the International Conference on Neural Information Processing Systems - Volume 1*, NIPS 2012, pp. 1097–1105, Curran Associates Inc., 2012.

[33] J. F. Golding, A. Mueller, and M. A. Gresty, "A motion sickness maximum around the 0.2 hz frequency range of horizontal translational oscillation.," *Aviation, space, and environmental medicine*, vol. 72, no. 3, pp. 188–192, 2001.

[34] 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, pp. 158–166, June 2016.

[35] S. Dupuy, A. Egges, V. Legendre, and P. Nugues, "Generating a 3D simulation of a car accident from a written description in natural language: The CarSim system," in *Proceedings of the Workshop on Temporal and Spatial Information Processing - Volume 13*, TASIP 2001, pp. 1–8, Association for Computational Linguistics, 2001.

[36] 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*, Lecture Notes in Computer Science, pp. 191–206, Springer, Cham, Oct. 2016.

[37] 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 *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA 2017, pp. 1443–1450, 2017.

[38] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, "Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, ITSC 2016, pp. 1470–1475, 2016.

[39] S. Artzi, S. Kim, and M. D. Ernst, "Recrash: Making software failures reproducible by preserving object states," in *Proceedings of the European Conference on Object-Oriented Programming*, ECOOP 2008, pp. 542–565, 2008.

[40] W. Jin and A. Orso, "BugRedux: Reproducing field failures for in-house debugging," in *Proceedings of the International Conference on Software Engineering*, ICSE 2012, pp. 474–484, 2012.

[41] J. Roeßler, A. Zeller, G. Fraser, C. Zamfir, and G. Candea, "Reconstructing core dumps," in *Proceedings of International Conference on Software Testing, Verification and Validation*, ICST 2013, pp. 114–123, 2013.

[42] M. Soltani, A. Panichella, and A. van Deursen, "A guided genetic algorithm for automated crash reproduction," in *Proceedings of the International Conference on Software Engineering*, ICSE 2017, pp. 209–220, 2017.

[43] M. Fazzini, M. Prammer, M. d'Amorim, and A. Orso, "Automatically translating bug reports into test cases for mobile apps," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, pp. 141–152, 2018.

[44] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the International Symposium on Software Testing and Analysis*, ISSTA 2015, pp. 385–396, 2015.

[45] C. Wang, F. Pastore, and L. Briand, "Automated generation of constraints from use case specifications to support system testing," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, ICST 2018, pp. 23–33, 2018.

[46] A. Armand, D. Filliat, and J. Ibañez-Guzman, "Ontology-based context awareness for driving assistance systems," in *Proceedings of the IEEE Intelligent Vehicles Symposium Proceedings*, IV 2014, pp. 227–233, 2014.

[47] A. Armand, *Situation understanding and risk assessment framework for preventive driver assistance*. PhD Thesis, Université Paris-Saclay, May 2016.

[48] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the International Conference on Software Engineering*, ICSE 2018, pp. 303–314, 2018.

[49] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic autonomous driving system testing," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE 2018, 2018 - To Appear.

[50] J. Eggert, F. Damerow, and S. Klingelschmitt, "The foresighted driver model," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, IV 2015, pp. 322–329, 2015.

[51] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Search-based test case generation for cyber-physical systems," in *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2017, 2017.

[52] A. Rizaldi and M. Althoff, "Formalising Traffic Rules for Accountability of Autonomous Vehicles," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, ICTS 2015, pp. 1658–1665, 2015.