Generating Effective Test Cases for Self-Driving Cars from Police Reports

Alessio Gambi
alessio.gambi@uni-passau.de
University of Passau
Passau, Germany

Tri Huynh
huynhminhtri00@gmail.com
Saarland University/CISPA
Saarbruecken, Germany

Gordon Fraser
gordon.fraser@uni-passau.de
University of Passau
Passau, Germany

ABSTRACT

Autonomous driving carries the promise to drastically reduce the number of car accidents; however, recently reported fatal crashes involving self-driving cars show that such an important goal is not yet achieved. This calls for better testing of the software controlling self-driving cars, which is difficult because it requires producing challenging driving scenarios. To better test self-driving car software, we propose to specifically test car crash scenarios, which are *critical* par excellence. Since real car crashes are difficult to test in field operation, we recreate them as physically accurate simulations in an environment that can be used for testing self-driving car software. To cope with the scarcity of sensory data collected during real car crashes which does not enable a full reproduction, we extract the information to recreate real car crashes from the *police reports* which document them. Our extensive evaluation, consisting of a user study involving 34 participants and a quantitative analysis of the quality of the generated tests, shows that we can generate accurate simulations of car crashes in a matter of minutes. Compared to tests which implement non critical driving scenarios, our tests effectively stressed the test subject in different ways and exposed several shortcomings in its implementation.

CCS CONCEPTS

• **Software and its engineering** → **Virtual worlds training simulations**; **Software verification and validation**.

KEYWORDS

automatic test generation, natural language processing, procedural content generation, self-driving cars

ACM Reference Format:

Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating Effective Test Cases for Self-Driving Cars from Police Reports. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3338906.3338942

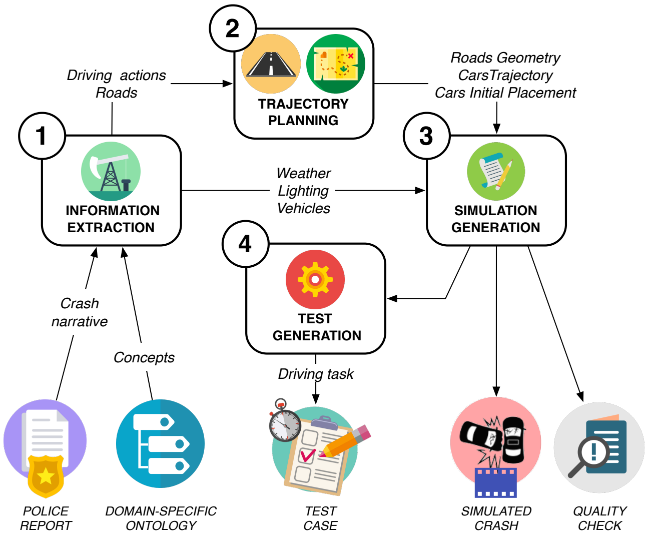

Figure 1: AC3R Overview

The figure illustrates the main steps which compose the proposed approach for automatically generating simulation-based tests from police reports of car crashes.

1 INTRODUCTION

Autonomous driving is the industry's promise to eliminate most accidents [9]; however, crashes involving self-driving cars, some of which were fatal for drivers [26] and pedestrians [39], testify that currently autonomous driving is not as safe as promoted. In many cases, the reason is defective software, which clearly indicates the need for a better approach for testing self-driving car software.

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. This is not only impractical and dangerous, but also ineffective, as it misses the most interesting, *critical* testing scenarios [22]. An alternative lies in *virtual tests*, where computer simulations are used to challenge self-driving car software [8, 44, 47]. While this provides the opportunity to automatically generate tests (e.g., [1, 12, 23, 41]), 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 (Automatic Crash Constructor from Crash Reports), 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

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3338942

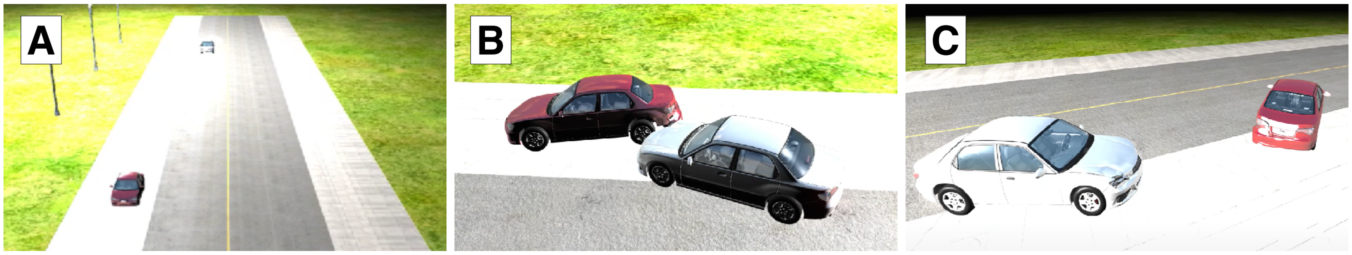

Figure 2: Key frames of the simulation generated by AC3R from the working example (NHTSA report #2005011269283). Frame A shows the virtual cars in their initial position; frame B shows the impact; frame C shows the crash aftermath. Video recordings of further simulations generated by AC3R are included in AC3R's demonstration [18].

data collected by the cars during real crashes is rarely available and detailed enough to enable a full reproduction [15]. Therefore, AC3R adopts a different strategy and extracts the information to recreate the car crashes from the *police reports* documenting 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 (e.g., [31]), cover many types of accidents, and are stored in large, usually publicly accessible, databases (e.g., [30]).

The technical challenge is that police reports are written in natural language, hence only partially structured. 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, and automatically generating code which re-enacts the car crashes as computer simulations, suitable as test cases (see Figure 1).

In detail, the contributions of this paper are as follows:

- We present AC3R, the first approach for testing self-driving car software using automatically reconstructed car crashes.
- We evaluate the accuracy of the reconstructed car crashes by means of an empirical user study involving 34 participants.
- We evaluate the quality of the generated test cases by testing a state-of-art vision-based self-driving car software system.

Our extensive evaluation shows that AC3R is efficient and generates simulations that accurately reproduce several types of crashes in minutes. Additionally, the tests derived from such simulations exposed faults in the self-driving car software we used as test subject. To favor the replication of our studies as well as to enable further research on this topic we release AC3R's source code, evaluation data, and instructions to replicate the experiments at:

https://github.com/SoftLegend/AC3R-Demo

2 AUTOMATICALLY RECONSTRUCTING AND SIMULATING REAL CAR CRASHES

AC3R adopts a four-step approach to derive simulation-based tests from reconstructed car crashes from police reports (Figure 1):

(1) **Information extraction**: AC3R relies on NLP to extract information about the crash contributing factors from the police reports, and maps concepts in the text to details of the crash scenario by means of a domain-specific ontology.
(2) **Trajectory planning**: 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**: AC3R generates code which implements the dynamics of the car crash in a driving simulator as well as runtime checks to determine when the accuracy of the simulations is not satisfactory.
(4) **Test generation**: AC3R derives test cases from the simulations by including test oracles, which check if the *ego-car*, i.e, the self-driving car under test, reaches its final expected position while avoiding 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 [30]. Figure 2 shows the key frames of the simulation which AC3R generated from the following 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."*

2.1 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. During information extraction, AC3R incrementally parses the narrative of the car crash and accumulates the information about weather, lighting, roads, and vehicles, into a number of data structures which form the abstract car crash.

2.1.1 Grammatical Dependencies Analysis. AC3R computes the *grammatical dependencies* between each pair of words in each statement of the crash narrative using the Stanford NLP suite [11]. 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. From those dependencies, AC3R concludes that the weather was cloudy and the lighting was dark. The number of grammatical dependencies might be large, but not all of them are essential for reconstructing car crashes, hence AC3R discards unnecessary ones. For example, grammatical dependencies such as punctuations (`punct`) are non-essential and discarded.

2.1.2 Matching the Domain-Specific Ontology. AC3R tries to match the (stemmed [33]) words from the remaining grammatical dependencies with instances in a domain-specific ontology [36], that we developed taking inspiration by Armand et al. [3]. Our ontology contains about 35 concepts describing the environment (weather, lighting, and roads), the traffic participants and their actions (e.g., movements), as well as the accidents, organized in three levels:

- The *top level* provides the fundamental dichotomy between *environment_property* and car crashes *storyline*.
- The *middle level* identifies structurally complex objects which are relevant for reconstructing the car crashes, such as *traffic_participant*, *road*, and *weather*.
- The *bottom level* contains leaf classes, i.e., simple concepts which cannot be further decomposed, like *vehicle_type*, *pavement_material*, and *lighting*.

The domain-specific ontology connects classes at the same level and across levels to structure its knowledge; e.g., 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 bottom level. The domain-specific 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, named 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 domain-specific ontology maps *“Conditions ... were ... **dark**”* with the `dark` named instance of the *lighting* class, and defines for it a suitable value for configuring the simulator to achieve the result exemplified in Figure 2. In case of missing information, we use default values.

AC3R selects the data structures to update by looking at which ontology class defines the named instance. For example, from `nsubj(cloudy, Conditions)` AC3R matches the named instance `cloudy` of the *weather* class; therefore, it updates the data structure which stores the weather configuration values. Similarly, from `nsubj(dark,Conditions)` AC3R matches the `dark` named instance of the *lighting* class and updates the data structure storing lighting informations. In essence, finding a matching instance in the domain-specific ontology is the criterion which AC3R uses to distinguish relevant information, which are extracted, from irrelevant information, which are ignored.

For now, AC3R focuses only on the main crash contributing factors, i.e., lighting, weather, roads, and car movements; hence, it neglects other details, such as traffic signs and signals, that might nevertheless be important in some cases to accurately reconstruct the environment in which crashes happened. Additionally, we keep AC3R’s ontology illustrative and simple enough to cover most of 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 part of our ongoing work to improve the accuracy of the recreated car crashes.

2.1.3 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 uses the grammatical dependencies to build dependency chains that link the quantitative properties, with their measurement units, and their actual values. In our working example, from the sentence *“... [a] street with a **speed limit** of **25 mph**.”*, AC3R finds that `nsubj(mph,speed limit)` and `nummod(mph,25)` are related, hence it stores the value 25 mph as the speed limit of the road into the abstract car crash.

Parsing 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 might refer to roads *indirectly*, for example by using their direction (e.g., the “East/West” roadway). AC3R handles these cases by looking at the cardinality of nouns and updating the information of all the roads at once when the plural form is used, and by matching roads attributes by value otherwise.

2.1.4 Extracting Vehicle Properties. AC3R extracts the information about the vehicles involved in the crash and the driving actions which lead to it. Information about vehicles are extracted using the basic approach described in Section 2.1.2, but extracting information about 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”, from verbs, like “hit”, 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; the references to the striker and victim cars as well as the components damaged in the impact otherwise.

In our working example, from the sentence *“... the **front** of **V1 struck** the **back** of [V2] ... ”*, AC3R extracts the grammatical dependency `nsubj(struck, front)`, which refers to an active impact driving action and a damaged component, and links it to `nmod:of(front, vehicle1)`; hence, AC3R infers 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. By the end of this process, AC3R records that vehicle1 was driving, but suddenly left the road, and struck the back of the parked vehicle2.

2.2 Trajectory Planning

After extracting the information about the driving actions, AC3R computes the trajectory that the simulated cars must follow in order to recreate the impact; coincidentally, AC3R defines the geometry of the road which must comply with the direction of each car movement. Trajectories are implemented as sequences of *waypoints*, i.e., positions in the tridimensional space that the simulated cars must visit in order. By placing one waypoint at the impact location and sharing it with all the vehicles involved in the crash, AC3R ensures that the simulated vehicles can hit each other.

Figure 3 illustrates how AC3R plans the intercepting trajectories for the working example: (a) AC3R starts by placing the impact point on the origin of the reference coordinate space; we indicate the impact point with a cross-mark in the figure. (b) AC3R continues

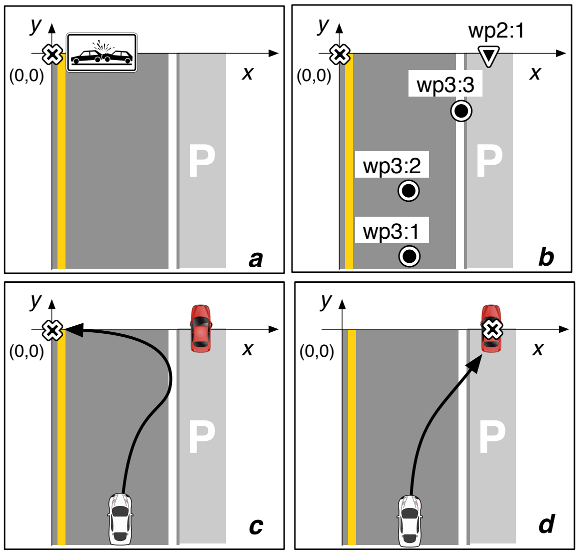

Figure 3: Trajectory planning using waypoints.

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. By suitably placing the waypoints, AC3R ensures that cars can reach the expect speed before impacting.

backwards by placing a new waypoint for each of the driving actions registered for each simulated vehicle. In our working example, vehicle1 drives, leaves the road, and hits vehicle2, while vehicle2 is parked; hence, AC3R places three waypoints for vehicle1 (circles) and one for vehicle2 (triangle). AC3R adds 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 the cars in the simulation. In Figure 3, the white and red cars respectively identify the initial positions of vehicle1 and vehicle2.

AC3R creates the trajectory of the striker car first (vehicle1 in the working example). For each driving action associated to the striker car, AC3R uses basic trigonometry and a simplified kinematics model to compute each segment of the trajectory. It does so assuming that the striker car travels at constant acceleration to reach the expected speed for the segment defined by two consecutive waypoints starting from the previously placed waypoint and moving in the specified direction. By placing the waypoints at a suitable distance, AC3R ensures that the striker has just enough room to reach the specified speed values before arriving at the next waypoint; hence, AC3R ensures not only that the simulation matches the police report but also limits the duration of its execution.

Initially, the impact point is placed on the center point of a straight road segment, which might be inaccurate if the victim car is parked or moves, or if the road is *curvy*. Therefore, (d) AC3R adjusts the position of the impact point and updates the road geometry. If the victim car is parked, like in our working example, AC3R moves the impact point to the side of the road and places the impact point directly on top of the victim car, ensuring this way the striker will hit the victim. If the victim car moves, first AC3R computes its trajectory in isolation; then, it adjusts the striker trajectory to hit the victim by computing an intercept trajectory.

2.3 Simulation Generation

AC3R relies on BeamNG.research [7], an extensible driving simulation which features accurate soft-body physics and realistic 3D textures of roads, vehicles, weather and lighting conditions. BeamNG.research 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.research enables external software to control the virtual cars, which is fundamental for testing self-driving car software [17].

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

AC3R generates code using a customized, template-based approach: For each of the relevant elements in the abstract car crash, AC3R produces a snippet of code which instantiates the corresponding object in the simulator. AC3R also generates a LUA script [19], which contains the dynamic elements of the simulations, including the code which triggers the movement of the cars.

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. So, after the simulation, AC3R can verify 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.

2.4 Test Generation

The simulations generated by AC3R contain the basic 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 check if the behavior of the ego-car is acceptable, and the ego-car cannot freely interact with the simulated environment. Therefore, we automatically derive system-level test cases from AC3R's output, in which the ego-car is free to choose a trajectory different to the one described in the police report in order to avoid the crash.

Such test cases need to meet some basic requirements; in particular, they shall (i) be relevant for the driving task at hand, i.e., avoid the crash; (ii) not alter the semantics of the original car crashes; and, (iii) be robust in the face of the stochastic nature of physics simulations. In order to meet these requirements, AC3R applies the following considerations when generating test cases.

2.4.1 *Choosing which Virtual Car the Ego-Car Drives.*

Car crashes usually involve more than one vehicle, hence AC3R may simulate more than one virtual car. In principle, the ego-car can control any of 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 legally parked car,"* 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. Furthermore, 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 would be pre-configured in the simulation, ends up driving off the road instead of crashing into the victim. This effectively changes the original semantics of the car crash from *"impacting a legally parked car"*into *"going off the road."*

2.4.2 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 safely reaching a goal waypoint, or its vicinity, placed behind the impact location computed by AC3R. To successfully complete the task and pass the test, the ego-car not only must reach the goal waypoint without crashing (safety), but also reach it within a given timeout (liveness). For safety, we strictly check that none of ego-car's components reports 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 non-ego-cars crash into each other. For liveness, 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.

2.4.3 False Positives. Intuitively, if the ego-car does not successfully complete a driving task this indicates a fault in its implementation. However, some combinations of ego and non-ego cars movements might result in *unavoidable* car crashes, or in car crashes with different semantics. For example, when a striker car arrives at an intersection before its victim, it might happen that the victim hits the striker effectively reversing the original semantics of the crash; similarly, it the victim's movements are belated, then the striker might be too close to the victim and cannot avoid hitting it. In those cases, failed tests would not expose faults in the ego-car, but represent *false positives*. To ameliorate this situation, one might define test preconditions on the movements of the various cars during the simulations such that those test executions which result in unavoidable crashes or change their semantics can be invalidated; however, false positive cannot be completely eliminated [6], and failed tests must be manually investigated to identify false positives.

3 EVALUATION

In order to empirically evaluate AC3R, we study how closely the simulations generated by AC3R match the descriptions from the police reports (RQ1, RQ2), how efficient the technique is when generating simulations (RQ3), and whether critical scenarios derived from the police reports are suitable as test scenarios (RQ4, RQ5).

In line with previous work on reconstructing car crashes for validating ADAS systems [15], we evaluate the effectiveness of our simulations "in-the-small":

RQ1. *How effective is AC3R at producing simulations with the expected impact?* Answering RQ1 lets us understand whether AC3R succeeds at generating simulations in which the virtual cars get damaged in a way which is compatible with the crash descriptions.

In line with previous work on visualizing car accidents [21] and reconstructing critical road segments [2], we 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 planning components of self-driving cars, like adaptive cruise controllers, object detectors, and lane keeping algorithms, would have prevented the car crashes in the first place.

In order to assess the practical applicability of our approach, we study how efficient AC3R is at analyzing police reports and generating the corresponding simulations:

RQ3. *How efficient is AC3R at generating crash simulations from police reports?* Answering RQ3 lets us understand if AC3R can be used within regular development activities, for instance to support explorative testing, or if it is better suited to support a one-off generation of tests.

Finally, we would like to validate our hypothesis that critical scenarios derived from police reports are useful for testing:

RQ4. *Do tests derived from critical scenarios exercise self-driving car software differently than their non-critical counterpart?*

RQ5. *Do tests derived from critical scenarios find more problems in self-driving car software than their non-critical counterpart?*

3.1 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) [30]. NHTSA reports follow the Model Minimum Uniform Crash Criteria (MMUCC) guidelines [31], which define the main crash contributing factors and state that crash-related events must be narrated in chronological order. To select the police reports, we partitioned the NHTSA police reports by crash type (e.g., "Frontal Impact", "Rear-End", "Straight Path", and so on), and removed the reports which violate AC3R working assumptions; next, we randomly sampled reports from each class. In summary, we selected 80 NHTSA reports which cover a wide range of environmental conditions, including day-light and night-time, roads composed of various types of road segments and intersections, and crash types.

To answer RQ1 – 3 we used AC3R to generate simulations of the critical scenarios described in the selected NHTSA reports. To answer RQ4 – 5, instead, we used AC3R to generate simulations of the critical scenarios described in the selected NHTSA reports as well as their non-critical counterpart; next, we derived test cases from both critical and non-critical scenarios and used them to test, *DeepDriving* [10], our test subject.

We generate non-critical scenarios from the critical ones by keeping the same overall organization of the simulation and the same definition of driving task, but relaxing specifically the hazards which cause the criticality. For example, in non-critical scenarios corresponding to rear-end crashes we slow down the striker car to allow for a more conservative stopping distance, while for cases where two cars collide at an intersection we stop the victim car just before entering the intersection.

DeepDriving [10] is a vision-based self-driving car software which uses a *convolutional neural network* (CNN) [25] to predict several driving affordance indicators[1] from the images captured by a forward facing camera attached to the front of car. Those indicators, along with the current speed of the car, are fed to a standard rule-based *driving controller* which computes the driving actions (i.e., steer, throttle, brake) to keep the car in the middle of lane, slow down before turning, and stop before impacting into other cars. In our evaluation, to enable the collection of coverage metrics we used a pre-trained implementation of DeepDriving's CNN based on TensorFlow[2] and we replaced the original driver controller written in C++ with an equivalent one written in Java.

For the evaluation, we executed AC3R, BeamNG.research, and DeepDriving 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.

3.2 RQ1: Effectiveness at Producing Simulations of the Impact

RQ1 investigates for how many police reports AC3R succeeds in producing a simulation, and 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", etc.), with the damaged components as reported by BeamNG.research during the simulation (e.g., "left headlight", "rear bumper", etc.).

We qualify the accuracy of the impact for each car by means of two binary variables which accounts for the damaged parts of the car ($\mathcal{P}$) and the car side ($\mathcal{S}$). Note that we consider the possibility of the police reports being underspecified. For example, if the simulator reports damage corresponding to the "front-left" of the car and the police report mentions "front-right", then we have a partial match since only $\mathcal{P}$ matches; however, if the police report mentions only "front", then a match of $\mathcal{P}$ results in a total match, regardless of $\mathcal{S}$. Having computed the values for $\mathcal{P}$ and $\mathcal{S}$, we label simulations as: *Unverified*, if AC3R was not able to process the police report; *Total match*, if both $\mathcal{P}$ and $\mathcal{S}$ match for all cars; *No match*, if $\mathcal{P}$ or $\mathcal{S}$ do not match for all cars; *Partial match* otherwise.

Table 1 reports the results of the simulations generated from the 80 police reports considered in this evaluation, by counting how many simulations fall into each category of the accuracy labels. AC3R was able to process more than half (62.6%) of the reports sampled from the NHTSA database; and, in the majority of those cases (78.0%) the simulations generated by AC3R reported the expected damage for all the virtual cars involved in the crash.

Table 1: Results from the impact analysis.

Label	Frequency	
	#	%
Total Match	39	48.8
Partially Match	7	8.8
No Match	4	5.0
Unverified	30	37.4
Total	80	100

Table 2: NHTSA police reports used in the user study (RQ1).

#	Case ID	Distinguishing Features
1	2005008586062	Day, 2-lane road, single car parked on curb
2	2005011269283	Night, 2-lane road, single car parked on curb
3	2005008586061	Day, 1-lane road, many cars parked on road
4*	–	Day, 1-lane road, car leaving its parking spot
5	2005002585724	Day, 2 moving cars, 2-way junction
6	2005004495041	Night, 2 moving cars, T-junction

*We obtained Case #4 by extending the description from Case #3 to include an additional element of complexity, i.e., the parked car leaving its spot.

We argue that the ability to produce accurate simulations for more than half of the reports is a substantial achievement considering the inherent limitations in the information extraction: AC3R currently adopts a very basic approach to NLP, which cannot handle all the corner cases of the narrative structures of the police reports; for example, AC3R cannot handle police reports which do not *strictly* follow the MMUCC guidelines or are semantically inconsistent. Furthermore, AC3R's implementation is not yet complete; for example, many concepts are missing from its domain-specific ontology, so, AC3R might lack the necessary elements to correctly build the simulations. The results also show that when AC3R produced the simulations, those were generally accurate, and only in few exceptional cases, the simulations did not result in the expected crash. In summary, we conclude that:

> *AC3R was able to fully reconstruct the impact between virtual cars for 57% of the considered police reports.*

3.3 RQ2: Accuracy of the Overall Simulations

RQ2 investigates how closely the environment reconstruction and the development of the whole car crashes 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 using an online survey 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. We considered a subset of the police reports in our dataset because the user tasks demand a non-negligible effort for the participants (7 minutes on average for each police report). Table 2 lists the police reports that we manually

[1] DeepDriving predicts 14 driving affordance indicators which include the position of the car in the lane, its heading angle, and the distance from the vehicles ahead of it.
[2] This version of DeepDriving's CNN achieved smaller error rates than the original implementation by Chen et al. as detailed at: https://bitbucket.org/Netzeband/deepdriving/wiki/DeepDrivingEvaluate

Table 3: 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 matches the description.

selected for this user study and summarizes their main distinguishing features, demonstrating that the selected reports cover many combinations of crash dynamics and environment configurations.

Table 3 lists the seven statements we used for evaluating the main aspects of the simulations. These include the general development of the simulation (#1), the setup and movement of virtual cars (#2 and #3), and different aspects of the crash (#4 – #6) and the environment (#7) reconstruction. For each statement, we used a 5-point Likert scale [27], 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 optional, short textual answers. To increase the robustness of our results towards the occurrence of random responses, we also included a *"Don't know/Don't want to answer"* category in addition to the five categories of the Likert scale.

A pilot study revealed that on average it took responders about 7 minutes to evaluate a single case. Therefore, we asked responders to evaluate only three cases to balance the trade-odd between the effort spent by the them for completing the survey and the benefit of collecting more data for our analysis. Responders participated anonymously and voluntarily, i.e., we did not use any monetary incentives or prizes 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 lets 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 practitioners, BeamNG developers, and the authors' personal contacts.

A total of 34 responders (response rate = 42.5%) completed the survey. Responders are aged between 18 to over 40, have decent technical background, 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. The 34 responders provided a total of 714 responses and 240 optional comments. Figure 4 visualizes the responses aggregated for each simulation generated from the reports listed in Table 2 to draw conclusions about the generality of our approach. Figure 5, instead, visualizes the responses aggregated for each statement in Table 3 to assess how accurately AC3R handles the different aspects of crashes reconstruction. 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.

Overall, responders considered the reconstructed scenarios to be a good match with the descriptions across the seven statements. In Figure 4, we observe that responders largely agreed that AC3R accurately reconstructed cases #1 – #4, while it was less accurate for cases #5 and #6. In 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 reconstruction (Statement #7), the overall development of the crash (Statement #1), and the movement of virtual cars (Statement #3).

Analysis of the optional comments provides more insights into these responses. 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 that happened at traffic junctions, and it results from the incompleteness of our prototypical implementation. Regarding the overall development of the crash and the movement of the virtual cars, instead, 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 described. 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 the police reports lack details about the 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. In summary, we conclude that:

AC3R reconstructions of our diverse study cases accurately reflected the descriptions in the police reports.

3.4 RQ3: Efficiency of AC3R

RQ3 investigates the practical applicability of AC3R by measuring its efficiency at generating the simulations by measuring AC3R's execution time when processing police reports and generating simulations. Across five repetitions, it took AC3R 44 seconds on average for the one time initialization of the Stanford NLP libraries required for the information extraction, while it took around 1 second on average for reconstructing each simulation. For reference, we also

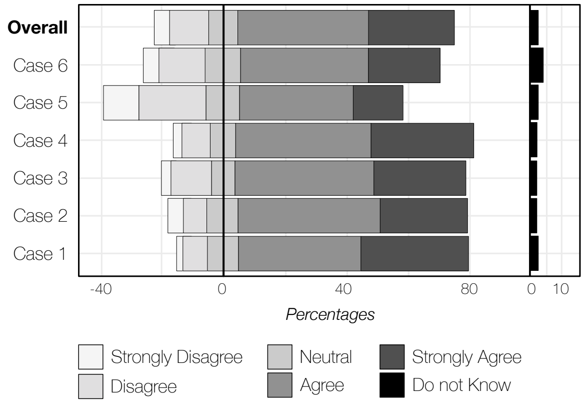

Figure 4: Aggregated responses per simulation to evaluate the generality of the approach (see Table 2).

computed the execution time for the simulations, which took on average 22 seconds. Anecdotally, according to an informal interview with the experts at BeamNG, manually generating each of these simulations would take up to two hours; hence, we conclude that:

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

3.5 RQ4: Coverage of AC3R Test Cases

RQ4 investigates whether the test cases derived from car crashes exercise the test subject in different ways than non-critical ones. We answer RQ4 by deriving test cases for critical and non-critical tests, comparing the coverage achieved by critical tests against the coverage achieved by their corresponding non-critical ones, and verifying that critical tests indeed cover targets which are not covered by the non-critical tests.

Specifically, we considered only the 39 test cases for which AC3R achieved the best accuracy (see Table 1), and configured DeepDriving to drive the *striker* car, as described in Section 2.4. We compute traditional code coverage metrics, i.e., branch coverage, on DeepDriving's driver controller. This gives us an intuition about how many behaviors of the controller are exercised by the critical tests but not by the non-critical ones. Additionally, we compute neuron coverage from DeepDriving's convolutional neural network to have an intuition also on how critical tests exercise different neurons comprising the network compared to non-critical tests.

As neuron coverage criterion we use the *k-multisection neuron coverage* (k-MNC) introduced by Ma et al. [28]. k-MNC measures how thoroughly each test covers the possible outputs of neurons by splitting neurons' *expected range of operation* into *k* sections and counting how many of such regions are covered during the test execution. We set the value of *k* to 10 under the considerations that, smaller values of *k* might hide the subtle differences in behaviors triggered by the tests, which are typical in neural networks (i.e., adversarial tests), while larger values of *k* might result in an extremely sensitive criterion misjudging similar behaviors as significantly different. To account for the fact that neurons might output values outside their expected range, we also include the *corner case regions* [28]. The computation of k-MNC requires the definition of the expected range of operation, and DeepDriving provides this information only for the neurons in the deepest layer of the CNN, i.e., the layer producing the network output. Hence, hereafter we consider only the coverage of those "output neurons."

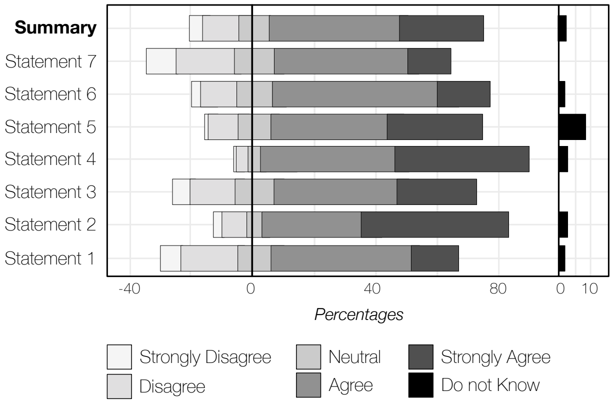

Figure 5: Aggregated responses per statement to evaluate the accuracy of reconstructed car crashes (see Table 3).

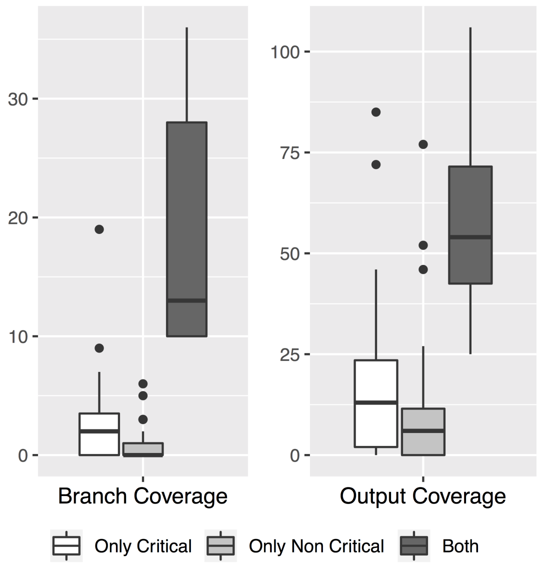

Figure 6: Code coverage

Figure 6 reports the differences in the code and neuron coverage metrics between critical and non-critical tests for DeepDriving's driver controller class (left plot) and CNN (right plot). In the plots, white bars refer to targets covered *only by critical tests*, i.e., the portion of the coverage targets covered by critical tests but not covered by non-critical tests; conversely, light gray bars identify the portion of targets covered *only by non-critical tests*. For completeness, we also report the portion of targets covered by *both* critical and non-critical tests as dark gray bars.

From Figure 6 we observe that both critical and non-critical tests covered a number of common branches (c.a. 13 on average). Critical tests consistently covered several branches which were not covered by non-critical tests (c.a. 2.5 on average), while non-critical tests did not generally cover branches which were not yet covered by the critical ones. Occasionally, critical tests covered larger numbers of branches (i.e., 9 and 19) which non-critical ones missed. We also observe that critical and non-critical tests cover at least a different sections for most (11 out of 14) of the output neurons. In particular, differences appear in how DeepDriving CNN perceives vehicles ahead of the ego-car and its position on the road.

The first observation suggests that critical tests might lead DeepDriving to take different decisions in case of critical and non-critical scenarios, while the second observation suggests that those different decisions might have been caused by different placements and movements of victim cars, which, in turn, led DeepDriving to drive the ego-car in different ways. Given these results, we conclude that:

> *The critical tests generated by AC3R exercised DeepDriving in different ways than the corresponding non-critical tests.*

3.6 RQ5: Effectiveness of AC3R Test Cases

RQ5 investigates the ability of the test cases derived from the simulations generated by AC3R to expose problems in self-driving car software. To answer RQ5, we executed the same tests used in RQ4 but count how many tests passed and failed. We manually inspected all the failed tests in order to check if they correspond to actual problems or false positives. Additionally, to assess the quality of DeepDriving's CNN predictions, we recorded the values of the driving affordance indicators predicted by DeepDriving's CNN and the corresponding ground truth values provided by the simulator during tests execution.

Figure 7 summarizes the results as bar plots which report for the critical (white) and non-critical (dark gray) cases the number of times DeepDriving *did not avoid the crash* (left), *did not reach the destination* (middle), and *safely reached the destination* (right).

From the results in Figure 7, we observe that overall the tests which implement critical scenarios failed twice as often as their non-critical counterpart (16 failed critical tests vs. 8 failed non-critical tests). Notably, critical tests failed more often because DeepDriving did not avoid the crash (12 failed critical tests vs. 3 failed non-critical test), while slightly more non-critical tests failed because DeepDriving did not reach the target location (4 failed critical tests vs. 5 non-failed critical tests).

Manual inspection of the failed critical tests reveals that in three cases the ego-car faced unavoidable crashes, and in three cases the goal waypoints were misplaced such that DeepDriving missed them. Notably, in eight cases DeepDriving failed to avoid preventable crashes, and in an additional case the ego-car could not find the travel lane and missed the correctly placed goal waypoint.

By comparing DeepDriving's CNN predictions with the ground truth data from the simulator, we notice that in all the critical tests DeepDriving largely miscalculated the distance between the ego-car and the other cars in front of it. We also notice that in several occasions (i.e., 16 out 39 cases) DeepDriving wrongly interpreted the road markings, which sometimes caused DeepDriving to drive the ego-car against the direction of traffic.

To assess the impact of CNN mis-predictions on the ability of DeepDriving's driving controller to handle critical situations, we repeated the execution of the 8 tests in which the ego-car could have avoided the crash by directly feeding the driving controller with the ground truth data collected from the simulator. Even given perfect knowledge, the controller was not able to pass all the tests, and in half of the cases the ego-car was either stuck behind the victim or crashed into it.

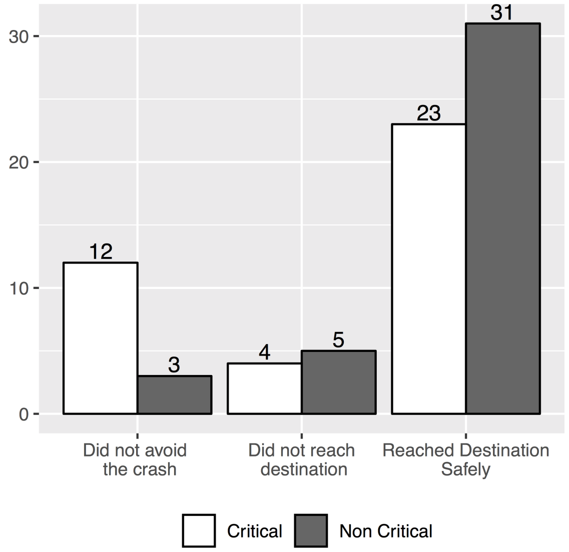

Figure 7: Test case effectiveness

The first observation suggests that the tests derived from critical scenarios effectively put the test subject in more hazardous situations than non-critical ones; and, since those situations are harder to handle for DeepDriving, critical tests fail more often. The second observation suggests that the CNN is faulty. Because DeepDriving's CNN was not trained with a diversified set of images including driving at night or in bright light [40, 45], it poorly predicts the driving affordance indicators in those cases. The third observation, instead, suggests that the controller is also faulty. Despite being able to correctly keep the lane and follow the victim, DeepDriving's driver controller was not be able to avoid other cars. In light of these results, we positively answer RQ5:

> *The test cases we derived from the real car crashes effectively exposed faults in both DeepDriving's CNN and driving controller.*

3.7 Threats to Validity

Internal validity. To ensure that our integration of DeepDriving into BeamNG.research is correct, we used the author's guidelines to set it up, manually tested the integration, and validated DeepDriving by having it successfully drive along randomly generated roads. To ensure that our Java implementation behaves equivalently to the original C++ implementation, we checked that the control actions computed by both driving controllers match. While AC3R does not currently recreate all elements (e.g., traffic signs), all required elements specified by DeepDriving, are included in the reconstruction; that is, the simulations generated by AC3R match DeepDriving's requirements. To avoid bias in the user survey, we anonymized it and assigned tasks to participants randomly.

External validity. We manually selected a small number of police reports from the NHTSA database to conduct the user study, because the elaborated tasks which that study requires do not easily scale; however, we ensured that the selected reports cover different crash types, lighting conditions, road geometries, number of cars, and car movements. Notably, the specific four types of crashes considered in the user study cover more than 50% of the entire dataset. Our user study, like any empirical investigation, involved a limited number of participants, so results might not generalize. We tried to diversify respondents by involving participants of different ages, from industry and academia, and with driving and decent technical experience. We only used a single self-driving car software system for evaluation, because most self-driving car software is not publicly available, and because the available implementations, which are used in related work [24, 28, 32, 40, 45], do not implement full-featured self-driving car software. While we therefore cannot say if effectiveness results generalize, our evaluation nevertheless demonstrates that AC3R is able to generate relevant test scenarios.

4 RELATED WORK

A central element of AC3R is the reproduction of real crashes for testing. There exists previous work on producing simulations of crashes, although not for the purpose of testing self-driving car software. Erbsmehl [15] recreated car crashes by replaying the sensory data collected during actual 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). CarSim [13] aims to generate simulations of car accidents from short textual descriptions. However, unlike AC3R, the objective is not to generate test cases, but to visualize crashes for humans to support understanding. For this, CarSim generates immutable animations by populating pre-defined scenarios, which are not suitable for deriving tests as they do not account for the interactions of a system under test. Furthermore, unlike AC3R, CarSim does not provide quality control, such as checking whether the animations are physically possible. Wang et al. [43], instead, use NLP and a class diagram with predefined OCL constraints to interpret use case specifications in the Restricted Use Case Modeling format for testing driver presence detection sensors according to the model-in-the-loop paradigm.

There are other approaches that aim to generate virtual tests. Sippl et al. [37] identify relevant test cases by mining large amounts of simulation data; Mullis et al. [29] find relevant test cases by identifying the performance boundaries of the system under test; Abdessalem et al. [1, 8] search for test cases which maximize an application-specific fitness function; Althoff and Lutz [2] search for critical cases using reachability analysis; and, Tuncali et al. [42] search for test cases which falsify safety properties of the cars. While these approaches aim to generate large numbers 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 [5, 20, 35, 38]. 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. [16], 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.

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 completely synthetic images or modifying real images which cause the convolutional neural networks to make poor predictions [24, 40, 46]. While these adversarial techniques require extensive knowledge about the internals of the convolutional neural networks to generate single images, AC3R is completely black-box and generates entire simulations, hence is more general.

5 CONCLUSIONS AND FUTURE WORK

In this paper we presented AC3R, a novel approach for efficiently generating computer simulations, which reproduce real car crashes, and test cases from police reports. Despite the simple nature of the generated simulations, our virtual tests can test self-driving cars under critical conditions, and they succeed at finding problems as our extensive evaluation showed. For testing, simple scenarios are even preferable: they are focused, better reproducible, and might be easier to understand than complex ones; furthermore, problems found using simple scenarios are most likely true positives since the self-driving cars are subjected to a lower level of noise.

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 aim to extend AC3R's knowledge base and information extraction for increasing the generality of the approach, including more details in the reconstructions (e.g., traffic, traffic signs), and simulating crashes aftermath. As identified by our user study this will improve the accuracy of our simulations. Future extensions of AC3R include:

- **Simultaneous localization and mapping** can improve the precision of the environment reconstruction using high definition maps.
- **Sophisticated traffic and drivers models** can replace the basic kinetics model used for trajectory planning, increasing the precision of the reconstruction dynamics [14].
- **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 [4].
- **Official traffic regulations** can be formalized into automatically verifiable formulae [34], which will provide more and better test oracles.

ACKNOWLEDGMENTS

This work is supported by EPSRC project EP/N023978/2.

REFERENCES

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

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

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

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

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

[6] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering 41*, 5 (May 2015), 507–525.

[7] BeamNG. BeamNG.research. https://beamng.gmbh/research/, 2018.

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

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

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

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

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

[13] Dupuy, S., Egges, A., Legendre, V., and Nugues, P. 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* (2001), TASIP 2001, Association for Computational Linguistics, pp. 1–8.

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

[15] Erbsmehl, C. 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* (2009), no. 09-0162 in ESV 2009, pp. 1–10.

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

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

[18] Huynh, T., Gambi, A., and Fraser, G. AC3R: automatically reconstructing car crashes from police reports. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019.* (2019), G. Mussbacher, J. M. Atlee, and T. Bultan, Eds., IEEE / ACM, pp. 31–34.

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

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

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

[22] Kalra, N., and Paddock, S. M. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice 94* (Dec. 2016), 182–193.

[23] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. Learning to drive in a day. *arXiv preprint arXiv:1807.00412* (2018).

[24] Kim, J., Feldt, R., and Yoo, S. Guiding deep learning system testing using surprise adequacy. *CoRR abs/1808.08444* (2018).

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

[26] Lambert, F. Understanding the fatal tesla accident on autopilot and the nhtsa probe. *Electrek* (2016).

[27] Likert, R. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[28] Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., and Wang, Y. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (New York, NY, USA, 2018), ASE 2018, ACM, pp. 120–131.

[29] Mullins, G. E., Stankiewicz, P. G., and Gupta, S. K. 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* (2017), ICRA 2017, pp. 1443–1450.

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

[31] 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.

[32] Pei, K., Cao, Y., Yang, J., and Jana, S. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 1–18.

[33] Porter, M. F. An algorithm for suffix stripping. *Program 14*, 3 (1980), 130–137.

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

[35] Roessler, J., Zeller, A., Fraser, G., Zamfir, C., and Candea, G. Reconstructing core dumps. In *Proceedings of International Conference on Software Testing, Verification and Validation* (2013), ICST 2013, pp. 114–123.

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

[37] Sippl, C., Bock, F., Wittmann, D., Altinger, H., and German, R. From Simulation Data to Test Cases for Fully Automated Driving and ADAS. In *Testing Software and Systems* (Oct. 2016), Lecture Notes in Computer Science, Springer, Cham, pp. 191–206.

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

[39] 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.

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

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

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

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

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

[45] Zhang, M., Zhang, Y., Zhang, L., Liu, C., and Khurshid, S. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (New York, NY, USA, 2018), ASE 2018, ACM, pp. 132–142.

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

[47] Zhao, D., and Peng, H. From the Lab to the Street: Solving the Challenge of Accelerating Automated Vehicle Testing. *arXiv preprint arXiv:1707.04792* (2017).