# Detecting Multi-sensor Fusion Errors in Advanced Driver-Assistance Systems

Ziyuan Zhong
ziyuan.zhong@columbia.edu
Columbia University
United States

Zhisheng Hu
zhishenghu@baidu.com
Baidu Security
United States

Shengjian Guo
sjguo@baidu.com
Baidu Security
United States

Xinyang Zhang
xinyangzhang@baidu.com
Baidu Security
United States

Zhenyu Zhong
edwardzhong@baidu.com
Baidu Security
United States

Baishakhi Ray
rayb@cs.columbia.edu
Columbia University
United States

## ABSTRACT

Advanced Driver-Assistance Systems (ADAS) have been thriving and widely deployed in recent years. In general, these systems receive sensor data, compute driving decisions, and output control signals to the vehicles. To smooth out the uncertainties brought by sensor outputs, they usually leverage **multi-sensor fusion** (MSF) to fuse the sensor outputs and produce a more reliable understanding of the surroundings. However, MSF cannot completely eliminate the uncertainties since it lacks the knowledge about which sensor provides the most accurate data and how to optimally integrate the data provided by the sensors. As a result, critical consequences might happen unexpectedly. In this work, we observed that the popular MSF methods in an industry-grade ADAS can mislead the car control and result in serious safety hazards. We define the failures (e.g., car crashes) caused by the faulty MSF as fusion errors and develop a novel evolutionary-based domain-specific search framework, *FusED*, for the efficient detection of fusion errors. We further apply causality analysis to show that the found fusion errors are indeed caused by the MSF method. We evaluate our framework on two widely used MSF methods in two driving environments. Experimental results show that *FusED* identifies more than 150 fusion errors. Finally, we provide several suggestions to improve the MSF methods we study.

## CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; **Software testing and debugging**.

## KEYWORDS

software testing, multi-sensor fusion, causal analysis, advanced driving assistance system

**ACM Reference Format:**
Ziyuan Zhong, Zhisheng Hu, Shengjian Guo, Xinyang Zhang, Zhenyu Zhong, and Baishakhi Ray. 2022. Detecting Multi-sensor Fusion Errors in Advanced Driver-Assistance Systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22), July 18–22, 2022, Virtual, South Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3533767.3534223

## 1 INTRODUCTION

Advanced Driver-Assistance Systems (ADAS) are human-machine systems that assist drivers in driving and parking functions and have been widely deployed on production passenger vehicles [6] (e.g. Tesla's AutoPilot and Comma Two's OPENPILOT [23]). Unlike the full automation promised by so-called self-driving cars, ADAS provides partial automation like adaptive cruise control, lane departure warning, etc., to promote a safe and effortless driving experience. Although ADAS are developed to increase road safety, they can malfunction and lead to critical consequences[1]. It is thus important to improve the reliability of ADAS.

A typical ADAS, as shown in Figure 1, takes inputs from a set of sensors (e.g., camera, radar, etc.) and outputs driving decisions to the controlled vehicle. It usually has a perception module that interprets the sensor data to understand the surroundings, a planning module that plans the vehicle's successive trajectory, and a control module that makes concrete actuator control signals to drive the vehicle. Oftentimes individual sensor data could be unreliable under various extreme environments. For example, a camera can fail miserably in a dark environment, in which a radar can function correctly. In contrast, a radar can miss some small moving objects due to its low resolution, while a camera usually provides precise measurements in such cases. To enable an ADAS to drive reliably in most environments, researchers have adopted complementary sensors and developed multi-sensor fusion (MSF) methods to aggregate the data from multiple sensors to model the environment more reliably. If one sensor fails, MSF can still work with other sensors to provide reliable information for the downstream modules and enable the ADAS to operate safely.

However, an MSF hardly knows which sensor output to rely on at each time step. Thus, it neither thoroughly eliminates the uncertainty nor always weighs more on the correct sensor data. This inherent flaw may introduce safety risks to the ADAS. In this paper, we study a popular commercial ADAS named OPENPILOT and show that sometimes its MSF can mistakenly prioritize faulty

*ISSTA '22, July 18–22, 2022, Virtual, South Korea*

ACM ISBN 978-1-4503-9379-9/22/07…$15.00
https://doi.org/10.1145/3533767.3534223

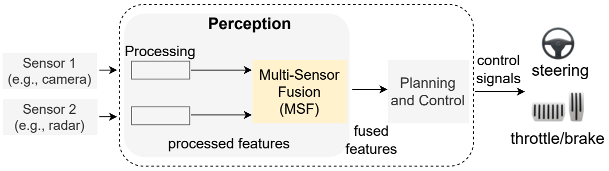


**Figure 1: The architecture of a typical ADAS system.**

sensor information over the correct ones. Such incorrect fusion logic can lead the vehicle to critical accidents. To this end, this paper focuses on automatically detecting accidents (i.e., collisions) that can occur due to incorrect fusion logic—we call such accidents as *fusion errors*.

Similar to existing ADAS testing [2–4], we resort to simulation rather than real-world testing as the latter is prohibitively expensive. A vehicle controlled by the ADAS, a.k.a. the *ego car*, drives through the scenario generated by a simulator. Here, the MSF logic of the ADAS is under test. The semantic validity of the generated scenarios are guaranteed by the usage of the simulator's traffic manager which controls other vehicles (not controlled by ADAS) to behave in a realistic way. Our aim is to simulate scenarios that facilitate fusion errors as detecting fusion errors even in a simulated environment is challenging.

**Challenges.** There are two main challenges in simulating and detecting fusion errors:

I. The failure (i.e., collision) cases in a commercial-grade ADAS are rare since it functions properly most of time. The failure cases caused by the fusion method are even more sparse since there can be other causes of failure like the malfunctions of all the sensors. Given testing an ADAS is costly, it is non-trivial to identify these fusion-induced failure cases within a limited time budget.

II. Even if we detect a failure, it is hard to conclude its root cause is an incorrect fusion logic. Employing simple differential testing (i.e., we simulate the whole driving scenario with alternative fusion logic and avoid the collision) cannot say with certainty that the root cause was the faulty fusion logic. This is because many uncertainties are involved in the simulation process—non-deterministic sensor outputs, random time delays between simulator and ADAS, etc.; reproducing the exact collision is non-trivial.

**Our Approach.** We treat a failure (i.e., collision) caused by the faulty fusion method as a fusion error. A reasonable assumption is that a *fusion fault* occurs as the fusion method chooses a wrong sensor output while a correct output from another sensor was available. Consequently, a *fusion error* usually takes place when (i) some *fusion faults* happen, and (ii) a *failure* (i.e., ego car collision) takes place. To detect fusion errors, we first use fuzz testing with objectives promoting the occurrence of many fusion faults and the resulting failure. If a failure happens, we further apply root cause analysis to filter out the failures that may not have been caused by faulty fusion logic. These two steps, as detailed below, are carefully designed and implemented with a tool, *FusED* (Fusion Error Detector), to address the challenges mentioned above.

*Step-I: Fuzzing.* To induce fusion errors, the simulator needs to generate scenarios that promote the fusion component to provide inaccurate prediction although the non-chosen sensor output provides accurate prediction, and lead the ego car to collision. In the driving automation testing domain, recent works leverage fuzz testing to simulate input scenarios in which an ego car runs and the fuzzer is optimized to search for failure-inducing scenarios [2, 21, 28, 45]. However, these methods treat the ego-car system as a black-box and ignore the attainable run-time information of the system. Inspired by the grey-box fuzzing of traditional software fuzzing literature [32], we propose an evolutionary algorithm-based fuzzing that utilizes the input and output information of the fusion component of the system. In particular, to promote the fusion component to make more fusion faults, we propose a novel objective function that maximizes the difference between the fusion component's prediction and the ground-truth, while minimizing the difference between the most accurate sensor's prediction and the ground-truth. Here, ground-truth is the actual relative location and relative speed of the leading vehicle w.r.t. the ego-car. To promote the ego car's crash, similar to previous works [2, 28, 45], we use an objective minimizing the ego car's distance to its leading vehicle. The two objectives synergistically promote finding scenarios that trigger fusion errors.

*Step-II: Root Cause Analysis.* To address challenge-II, i.e., to check whether the observed failure is indeed due to the fusion logic, we study if the failure still happens after choosing an alternate fusion logic in an otherwise identical simulation environment. Here, we intend to do a controlled study to measure the effect of faulty fusion logic. However, maintaining an identical setting is infeasible because of many uncertainties and randomness in the environment of simulator and controller. Thus, we rely on the theory of causal analysis. Based on the understanding of the studied ADAS and the simulator, we construct a causal graph, where graph nodes are all the variables that can influence the occurrence of a collision during a simulation, and the edges are links that show their influence with each other. We then intervene and change the fusion logic by keeping all the other nodes identical in the causal graph. Such intervention is applied by setting the communications between the simulator and ego-car deterministic and synchronous for all the simulations. To efficiently find a fusion method that can avoid the collision, we use a *best-sensor fusion* method, which always selects the sensor's output that is closest to the ground truth. If we no longer see the collision in this counterfactual world, we conclude that the root cause of the observed collision was incorrect fusion logic. Otherwise, we discard the failure observed during fuzzing. To further reduce double-counting the same fusion error, we propose a new counting metric based on the coverage of the ego car's location and speed trajectory during each simulation.

To the best of our knowledge, our technique is the first fuzzing method targeting the ADAS fusion component. In total, *FusED* has found more than 150 fusion errors. In summary, we make the following contributions:

- We define *fusion errors* and develop a novel grey-box fuzzing technique for efficiently revealing the *fusion errors* in ADAS.
- We analyze the causes of the *fusion errors* using causal analysis.

- We evaluate *FusED* in an industry-grade ADAS, and show that it can disclose safety issues.
- We propose suggestions to mitigate *fusion errors* and effectively reduce *fusion errors* in a preliminary study.

The source code of our tool and interesting findings are available at https://github.com/AIasd/FusED.

## 2 BACKGROUND: FUSION IN ADAS

The "Standard Road Motor Vehicle Driving Automation System Classification and Definition" [24] categorizes driving automation systems into six levels. Advanced Driver-*Assistance* Systems (ADAS) usually consists of levels 0 to 2, which only provides temporary intervention (e.g., Autonomous Emergency Braking (AEB)) or longitudianl/latitudinal control (e.g., Automated Lane Centering (ALC) and Adaptive Cruise Control (ACC)) while requiring the driver's attention all the time. In contrast, Automated Driving Systems (ADS) consist of levels 3 to 5, which allow the driver to not pay attention all the time. In this section, we introduce commonly used fusion methods and related errors for driving automation systems. In particular, we focus on Openpilot, a level2 industry-grade ADAS. However, we believe our approach can also generalize to ADS that use similar fusion methods [9, 39].

We next define the terminologies used later.

- A *driving environment* is a parameterized space where search during the fuzzing will be bounded.
- A *scenario* is a concrete instance in the driving environment.
- The *ego car* is the vehicle controlled by the ADAS under test.
- The *NPC (non-player character) vehicles* are the vehicles other than the ego car.
- The *leading vehicle* is the vehicle ahead of the ego car in the same lane.
- A high-fidelity *simulator* provides an end-to-end simulation environment for testing ADAS. It generates sensor data at regular intervals (from cameras, radar, etc.) that can be fed into the ADAS under test, and receives control signal from the ADAS to update the ego car in the simulated world.

### 2.1 Fusion in Driving Automation

Most industry-grade driving automation systems, including ADAS and ADS, leverage multi-sensor fusion (MSF) to avoid potential accidents caused by the failure of a single sensor [9, 14, 39]. MSF often works with camera and radar, camera and Lidar, or the combination of camera, radar, and Lidar. Yeong et al. [42] provide a survey on sensor fusion in autonomous vehicles. They categorize MSF into three primary types: high-level fusion (HLF), mid-level fusion (MLF), and low-level fusion (LLF). These MSFs differ in how the data from different sensors are combined. In HLF, each sensor independently carries out object detection or a tracking algorithm. The fusion is then conducted on the high-level object attributes of the environment (e.g., the relative positions of nearby vehicles) provided by each sensor and outputs aggregate object attributes to its downstream components. LLF fuses the sensor data at the lowest level of abstraction (raw data)[43]. MLF is an abstraction-level between HLF and LLF. It fuses features extracted from the sensor data, such as color information from images or location features of radar and LiDAR, and then conducts recognition and classification on them[29]. Among them, HLF is widely used in open-sourced commercial-grade ADAS [14] and ADS [9, 39] because of its simplicity. Thus, it is the focus of the current work. In particular, we conduct a carla simulator-based case study on an industry-grade ADAS, Openpilot, which uses an HLF for camera and radar.

### 2.2 Fusion in Openpilot

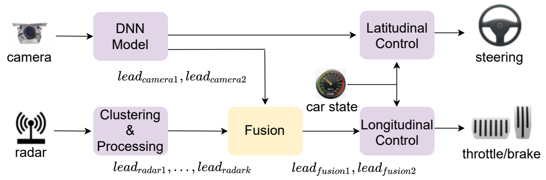
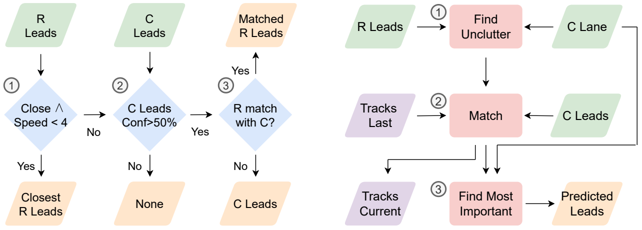


**Figure 2: The role of fusion in Openpilot.**

Figure 2 shows the the fusion component in Openpilot. It receives data about the leading vehicles from the camera processing component and the radar processing component. Each leading vehicle data, denoted as *lead*, consists of the relative speed, longitudinal, and latitudinal distances to the leading vehicle, and the prediction's confidence (only for camera). The fusion component aggregates all lead information from the upstream sensor processing modules and outputs an estimation to the longitudinal control component. Finally, the longitudinal control component outputs the decisions for throttle and brake to control the vehicle. Since the latitudinal control component only relies on camera data, we do not consider accidents due to the ego car driving out of the lane. Different fusion logics can be implemented. Here we studied Openpilot default one and a popular Kalman-Filter-based fusion method [30, 34].

*default: Heuristic Rule-based Fusion.* Figure 3a shows the logic flow of the Openpilot's fusion method default. It first checks if the ego car's speed is *low* ($ego_{speed} < 4$) and *close* to any leading vehicle (①). If so, the closest radar leads are returned. Otherwise, it checks if the *confidence* of any camera leads go beyond 50% (②). If not, leading vehicles will be considered non-existent. Otherwise, it checks if any radar leads match the camera leads (③). If so, the best-matching radar leads are returned. Otherwise, the camera leads are returned.



(a) default (b) mathworks

**Figure 3: Fusion logic of (a)Openpilot default and (b)mathworks. C denotes camera and R denotes radar. Green, orange, blue, red, and purple denote input, output, decision, processing, and stored data over generations, respectively.**

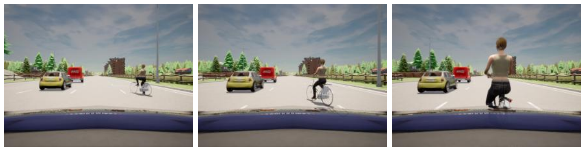
*mathworks: Kalman-Filter Based Fusion.* Figure 3b shows the logic of mathworks which is a popular fusion method from Mathwork

[34]. It starts with the camera-predicted lane to filter out cluttered (i.e. stationary outside the ego car's lane) radar leads in ①. Then, it groups together camera leads and uncluttered radar leads, and matches them with tracked objects from last generation in ②. Tracked objects are then updated. Finally, matched tracked objects within the current lane are ranked according to their relative longitudinal distances in ③ and the data of the closest two leads are returned.

## 2.3 Fusion Error & Motivating Example

A fusion error happens at the occurrence of the following:

i. Fusion logic makes some *fusion faults* as there is disagreement between different sensor outputs, and the underlying fusion logic trusts the incorrect one even when an alternate correct input is present, and
ii. A *failure*, i.e., a critical accident, takes place due to such faulty fusion method.



**(a)** $time_0$
**rel x:**
**camera:13.7m**
**(conf:0.1)**
**radar:19.2m**
**fusion:None**
**GT:None**

**(b)** $time_1$
**rel x:**
**camera:11.8m**
**(conf:13.5)**
**radar: 3.9m**
**fusion: None**
**GT: 2.9m**

**(c)** $time_2$
**collision happens.**

**Figure 4: A collision example. The ego car is the blue car whose front view has been shown.** *rel x*: **relative longitudinal distance of the ego car from the cycle. GT: the ground-truth value. The fusion fault and GT are highlighted in red and blue respectively.**

*Motivating Example.* Figure 4 shows an example where the ego car collides with a bicyclist cutting in. At $time_0$ (Figure 4a), no leading vehicle exists. At $time_1$ (Figure 4b), the bicyclist on the right trying to cut in. While the radar predicts that the lead is close (3.9 m) to the ground-truth (GT) value (2.9m), the camera ignores the bicyclist. The fusion component trusts the camera so the ego car does not slow down, and finally a collision occurs at $time_2$(Figure 4c). This example shows an accident caused by a wrong result from the fusion component. But how could the problem happen?

In the logic flow of the Openpilot's default fusion method (see Figure 3a), due to path (①) and (②), i.e., $\neg(ego_{speed} < 4 \wedge close)$ and ¬(camera confidence > 50%), no leading vehicle is considered existent at $time_1$ (i.e., Fusion output=None). Thus, the ego car accelerates until hitting the bicyclist.

# 3 OVERVIEW

We focus on finding fusion errors, i.e., failures caused by the faulty fusion method. We first define fusion fault and fusion error (see Section 3.1). Fusion methods are faulty when (i) two sensors' outputs differ significantly, and then (ii) the fusion logic (i.e., merging the sensor outputs) prioritizes the faulty outputs over the correct one. To simulate fusion errors, *FusED* first efficiently searches (a.k.a. fuzz) the given driving environment to find scenarios where the fusion method tends to prioritize faulty sensor outputs and thus, lead to failures (i.e., collisions) (see Section 3.2). *FusED* then changes the existing fusion logic with alternative ones and check whether the updated logic can avoid the simulated crashes (see Section 3.3). If a collision is avoidable with alternative fusion logic, *FusED* concludes that original fusion logic was erroneous. Finally, *FusED* reports the unique fusion errors, as described in Section 3.4.

## 3.1 Definitions

We first define the fusion fault of a fusion method $F$. Let, at a time step $t$, $F$ read $m$ sensor outputs $S_{t1}$,$S_{t2}$,...,$S_{tm}$ respectively, and outputs an aggregated prediction $F_t := F(S_{t1}, S_{t2}, ..., S_{tm})$. Let $GT_t$ denote the corresponding ground-truth value at the time step $t$. A correct fusion method should choose the sensor output closest to the ground truth to capture the most realistic situation. Thus, a fusion fault occurs if there is at least one sensor input, say $S_{tj}$ at time $t$, whose distance from $GT_t$ is less than the distance between fusion output $F_t$ and $GT_t$. To make the fault definition more tolerant to small errors, we further introduce an error tolerance threshold $th_{err}$.

**Definition 1.** *$F$ makes a **fusion fault** at a time step $t$ if*

$$\min_{j\in\{1,...,m\}} \mathbf{dist}(S_{tj}, GT_t) + th_{err} < \mathbf{dist}(F_t, GT_t),$$

One example of **dist** is $\mathbf{dist}(x, y) = ||x - y||_1$ which is simply the l1 distance. Note in this work we are not interested in benign fusion faults that cannot lead to critical consequences. Besides, it is difficult to attribute a failure to a particular fusion fault since a failure may appear as an effect of several fusion faults. Thus, we associate a failure to the underlying fusion method.

**Definition 2.** A **fusion error** occurs if the system under test using the fusion method fails due to faulty fusion method.

In this paper, we focus on the crashes of the ADAS to study fusion errors. As per its definition, a fusion error has the following two properties:

- *Failure-inducing*: A simulation should witness a failure of the system. In our context, the system is Openpilot and the failure is the ego car's crash. Besides, since only the longitudinal control module in Openpilot uses fusion, we only consider ego car's collision happening within the lane it follows.
- *Fusion-induced*: The failure should be caused by the used fusion method. In other words, if the rest of the system and environment behave as it is, and we had a correct implementation of the fusion method, the failure would not be observed.

Figure 5 illustrates such a fusion error. At time ①, a simulation starts and Openpilot is engaged. The simulation enters the pre-crash period (i.e., the $m$ seconds before an accident during which it starts to misbehave) at time ② and finally, a collision happens at ③. If a better fusion method is used from time ② onward, in the counter-factual world, no collision happens (at time ③' ).

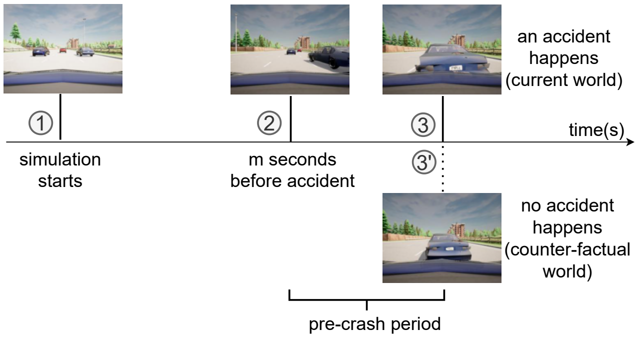

an accident happens (current world)
①
simulation starts
②
m seconds before accident
③
③'
time(s)
no accident happens (counter-factual world)
pre-crash period

**Figure 5: An illustration of pre-crash period and fusion error.**

## 3.2 Simulating Collisions with Fusion Fuzzing

To simulate fusion errors efficiently, we apply an evolutionary-based fuzzing algorithm that searches for the scenarios in which fusion errors are likely to happen. Since fusion errors have two properties: failure-inducing and fusion-induced (Section 3.1), we design the objective functions to optimize accordingly.

For capturing *failure-inducing* property, we adopt the safety potential objective used in [28]. It represents the distance between the ego car and the leading vehicle (subtracted by the ego car's minimum stopping distance)—minimizing it will facilitate the collision. We denote it as $\mathbf{F}_{\text{d}}(x)$ for the scenario $x$. To further promote collision, we introduce another boolean objective function ($\mathbf{F}_{\text{failure}}$) that is true only if a collision happens.

For the *fusion-induced* property, we define an objective $\mathbf{F}_{\text{fusion}}$ measuring fusion faults during an simulation, and maximize it. There can be many ways to define $\mathbf{F}_{\text{fusion}}$. Here, we use the number of time steps such that at each time step the fusion's output is far from the ground-truth and at least one sensor output is close to the ground-truth. Given that we use a simulated environment for testing, we can easily get the ground-truth lead information from the simulator. We present the details of $\mathbf{F}_{\text{fusion}}$ in Section 4.1. Putting the above objectives together, we obtain the following fitness function that our evolutionary fuzzer tries to optimize (here $c_i$s are coefficients):

$$\mathbf{F}(x) = c_{\text{failure}}\mathbf{F}_{\text{failure}}(x) + c_{\text{d}}\mathbf{F}_{\text{d}}(x) + c_{\text{fusion}}\mathbf{F}_{\text{fusion}}(x) \qquad (1)$$

## 3.3 Analyzing Root Causes of the Collisions

We next analyze the simulated failures (i.e., collisions) reported in previous step and check they are indeed *caused* by the incorrect fusion logic. The most intuitive approach to check this would be to simply replace the fusion method with another fusion method and check if the collision still happens. However, this approach has two issues. First, compared with the initial simulation, some unobserved influential factors (e.g., the communication delay between the simulator and OPENPILOT) might have changed. As a result, even if a collision does not occur with an alternative fusion logic, it might be due to the influence of other unobserved influential factors. Second, the alternative fusion logic chosen randomly may not be able to avoid the collision. Since simulation is costly, it is not possible to explore all the different logic (e.g., all the if-else branches in the fusion logic implementation Figure 3-a). Thus, we must choose the alternative fusion method carefully.

To address the first issue, we resort to the theory of causal analysis. In particular, we consider the fusion method used as the interested variable and the occurrence of a collision as the interested event. We then consider all other factors that can directly or indirectly influence the collision as well as their interactions based on domain knowledge, the understanding of the source code of OPENPILOT and the CARLA simulator, and simulation runtime behavior across multiple runs. The goal is to control all the factors that influence the collision and are not influenced by the fusion method to stay the same across the simulations. For those influential variables that cannot be controlled directly, we apply interventions on other variables such that the uncontrollable variable's influence on the collision is eliminated. For example, to eliminate the influence of the communication latency, which has been observed as the major uncontrollable influential variable, we set the communication configurations for OPENPILOT and simulator to be synchronous and deterministic. Assuming that all the influential variables are controlled, if the collision is avoided after the replacement in a counterfactual world, we can say the fusion method used is the actual cause.

To address the second issue, we define a fusion method called *best-sensor fusion*, which always selects the sensor prediction that is closest to the ground-truth as per **dist** in Definition 1. This fusion method provides the best prediction among the sensors. Consequently, it is reasonable to assume that it should help to avoid the collision if the collision was due to the fusion method used.

## 3.4 Counting Fusion Errors

We design the principles for counting distinct fusion errors in this section. Note that error counting in simulation-based testing remains an open challenge. Related works [2, 31] consider two errors being different if the scenarios are different. This definition tends to over-count similar errors when the search space is high-dimensional. Another approach manually judges errors with human efforts [28]. Such way is subjective and time-consuming when the number of errors grows up.

Inspired by the location trajectory coverage [21], we consider the ego car's state (i.e., location and speed) during the simulation rather than the input space variables or human judgement. We split the pieces of the lane that ego car drives on into $s$ intervals and the ego car's allowed speed range into $l$ intervals to get a two-dimensional coverage plane with dimensions $\mathbf{1}^{s\times l}$. During the simulation, the ego car's real-time location and speed are recorded. The recorded location-speed data points are then mapped to their corresponding "bins" on the coverage plane. Given all the data points mapped into the bins having the same road interval, their average speed is taken, the corresponding speed-road bin is considered "covered", and the corresponding field on the coverage plane is set 1. Note a simulation's final trajectory representation can have at most $s$ non-zero fields. We denote the trajectory vector associated with the simulation run for a specification $x$ to be $\mathbf{R}(x)$ and define:

**Definition 3.** Two fusion errors for the simulations runs on specifications $x_1$ and $x_2$ are considered distinct if $||\mathbf{R}(x_1) - \mathbf{R}(x_2)||_0 > 0$.

To demonstrate this error counting approach, we show two fusion errors with different trajectories in Figure 6. In both Figure 4

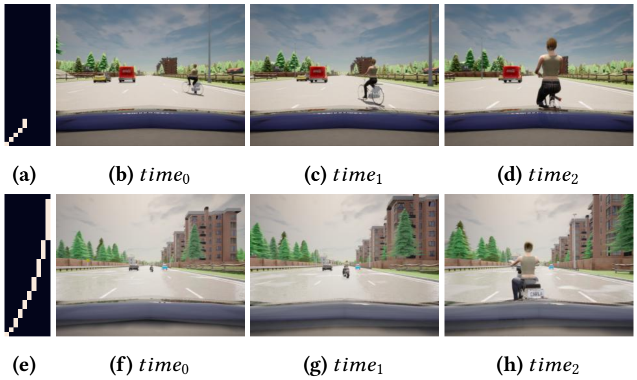


**Figure 6: Examples of two fusion errors with different trajectories. (a) and (e) show speed-location coverage where the x-axis is speed and y axis is the road interval.**

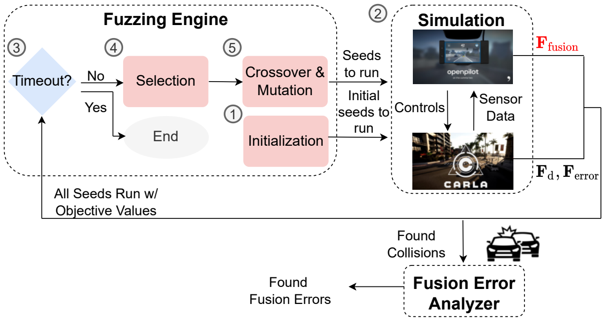


**Figure 7: The overall workflow of *FusED*.**

and the first row of Figure 6, the ego car hits a bicyclist cutting in from the right lane. The difference is only that the yellow car on the left lane has different behaviors across the two runs. However, the yellow car does not influence the ego car's behavior. Hence, the two simulation runs have the same trajectory coverage (ref. Figure 6a). By contrast, the other fusion error on the second row of Figure 6 has a different trajectory (ref. Figure 6e) that the ego car in high speed collides with a motorcycle at a location close to the destination. This example illustrates the necessity of counting fusion errors upon Definition 3.

## 4 *FUSED* METHODOLOGY

In this section, we introduce *FusED*, our automated framework for fusion errors detection. Figure 7 shows a high-level workflow of *FusED*. It consists of three major components: the fuzzing engine, the simulation, and the fusion error analyzer. The fuzzer runs for predefined rounds of generations. At each generation, it feeds generated scenarios (a.k.a. *seeds*) into the simulation. In a simulation, at each time step, the CARLA simulator supplies the sensor data of the current scene to OPENPILOT. After OPENPILOT sends back its control commands, the scene in CARLA updates. After the simulations for all the seeds at the current generation have been run, the seeds along with their objective values in the simulations are returned as feedback to the fuzzer. Besides, all the collision scenarios are recorded. The fuzzer then leverages the feedback to generate new seeds in the execution of the next generation. After the fuzzing process ends, all the collision scenarios are rerun with the best-sensor fusion in the counterfactual world. The scenarios that avoid the collision are reported as fusion errors.

### 4.1 Fuzzing Algorithm

*FusED* aims at maximizing the number of found fusion errors within a given time budget. The search space is high-dimensional and the simulation execution is costly. Evolutionary-based search algorithms have been shown effective in such situation[28, 45]. We adopt an evolutionary-based search algorithm with a domain-specific fitness function (defined in Section 3.2) promoting fusion errors finding. We denote our method as GA-FUSION.

The fuzzer tries to minimize a fitness function over generations. At the beginning, random seeds are sampled from the search space and fed into the simulation, as shown by ① in Figure 7. In ②, the simulation then runs OPENPILOT in CARLA with the supplied scenarios. The violations found are recorded and the seeds with the objective values are returned to the fuzzer accordingly. If the whole execution runs timeout, the fuzzing procedure ends (③). Otherwise, seeds are ranked based on their objective values for further *selection* (④). The fuzzer performs *crossover and mutation* operations among the selected seeds to generate new seeds (⑤) for the simulation. The steps ②-⑤ repeat until reaching the time threshold.

We next provide details for the selection step, the crossover & mutation step, and $F_{\text{fusion}}$, which is the objective promoting the occurrence of more fusion faults (see Section 3.2).

**Selection.** We use binary tournament selection, which has shown effectiveness in previous ADAS testing works [2]. For each parent candidate seed, the selection method creates two duplicates and randomly pairs up all the parent candidate seed duplicates. Each pair's winner is chosen based on their fitness function values. The winners are then randomly paired up to serve as the selected parents for the following crossover step.

**Crossover & Mutation.** We adopt the simulated binary crossover [7] following the approach in [2]. We set the distribution index $\eta = 5$ and probability=0.8 to promote diversity of the offspring. Further, we apply polynomial mutation to each discrete and continuous variable with mutation rate set to $\frac{5}{k}$, where $k$ is the number of variables per instance, and the mutation magnitude $\eta_m = 5$ to promote larger mutations.

**Details of $\mathbf{F}_{\text{fusion}}$.** $\mathbf{F}_{\text{fusion}}$ is defined as the percentage of the number of frames in which the fusion predicted lead having a large deviation from the ground-truth lead, while at least one predicted lead from upstream sensor processing modules is close to the ground-truth lead.

*Metric Function.* In order to quantify "a large deviation" and "close to", we first define the metric function $\mathbf{dist}(\cdot) : \mathbb{R}^{|\mathbf{D}|} \times \mathbb{R}^{|\mathbf{D}|} \to \mathbb{R}$ to measure the difference between two leads, where $\mathbf{D}$ is a set of indices of the different dimensions of a lead as defined in Section 2.2 (i.e., relative longitudinal distance, relative latitudinal distance, and relative speed). It takes in the two lead and outputs their distance. The metric function can be set to any reasonable metric, e.g., l1 norm of the two lead's difference. In the current work,

$$\mathbf{dist}(\hat{y}, y) := \sum_{j \in \mathbf{D}} \mathbb{1}[|\hat{y}_j - y_j| > th_j], \tag{2}$$

where $\mathbb{1}[\cdot]$ is a mapping of a condition's truth value to a numeric value in $\{0,1\}$. The function $\mathbf{dist}(\cdot,\cdot)$ thus counts the number of dimensions of the two leads ($\hat{y}$ and $y$) that differ more than a corresponding error threshold $th_j$. The threshold $th_j$ is set to 4m, 1m, and 2.5m/s for relative longitudinal distance, relative latitudinal distance, and relative speed, respectively. They are chosen based on domain knowledge. In particular, the length and width of a vehicle are roughly 4m and 1.5m and the default step size of Openpilot's cruise speed is roughly 2.5m/s. We next formally define $F_{\text{fusion}}$.

*Definition of $F_{fusion}$.* Given a scenario $x$,

$$\mathbf{F}_{\text{fusion}}(x) := \frac{1}{|\mathbf{P}|}\sum_{t\in\mathbf{P}} \mathbb{1}[\exists\ k \in \mathbf{K} \text{ s.t. } \mathbf{dist}(\hat{s}_{tk}, y_t) \le th \quad \text{and } \mathbf{dist}(\hat{f}_t, y_t) > th], \tag{3}$$

where $\mathbf{P}$ is a set of indices of the frames during the pre-crash period, $\mathbf{K}$ is a set of indices of the sensors (camera and radar in our case), $\hat{s}_{tk}$ is a predicted lead by sensor $k$ at time frame $t$, $\hat{f}_t$ is the predicted lead by the fusion component, $y_t$ is the ground-truth lead, and $th$ is a distance threshold. $th$ can be set to any non-negative values. In the current work, we set $th$ to 0. This implies that for a time frame $i$ to be counted, the fusion predicted lead $\hat{y}_i$ must violate at least one dimension and at least one sensor's predicted lead $\hat{s}_{ik}$ must not violate any of the three dimensions.

## 4.2 Root Cause Analysis

This step analyzes the fusion errors found by the fuzzer to confirm the incorrect fusion logic causes them. As described in Section 3.3, we leverage causal analysis to find the root cause of the failures, and if fusion logic is not the reason behind a failure, we filter it out.

***Problem Formulation.*** In causality analysis, the world is described by variables in the system and their causal dependencies. Some variables may have a causal influence on others. This can be represented by a Graphical Model [36], as shown in Figure 8, where the graph nodes represent the variables, and the edges connect the nodes that are causally linked with each other. For example, the test scenario should influence the occurrence of a collision. In a scenario involving many NPC vehicles, Openpilot is more likely to crash. The variables are typically split into two sets: *the exogenous variables* ($U$), whose values are determined by factors outside the model, and *the endogenous variables* ($V$), whose values are ultimately determined by the exogenous variables.

In our context, we define $\overrightarrow{X}$ to be the fusion method, $\overrightarrow{Y}$ to be a boolean variable representing the occurrence of a collision, and $\phi = \overrightarrow{Y}$. $\overrightarrow{Z}$ is the union of $\overrightarrow{X}$ and $\overrightarrow{Y}$. $\overrightarrow{W}$ is the complement of $\overrightarrow{Z}$ in $V$. Following the definition of actual cause in [18],

**Definition 4.** Given we know a collision ($\phi = True$) happens when a fusion method is used ($\overrightarrow{X} = \overrightarrow{x}$), the fusion method is an **actual cause** of a collision if: when another fusion method is used ($\overrightarrow{X} = \overrightarrow{x'}$), and all other endogenous variables (which influence the collision and are not influenced by the fusion method) are kept the same as in the original collision scenario ($\overrightarrow{W} = \overrightarrow{w}$), the collision can be avoided ($\phi = False$).

The details of the justification for this definition can be found in Appendix A in the extended version [44]. We use this definition as the basis to check if a found collision is a fusion error (i.e., if the used fusion method is the actual cause of the collision). In order to use it in practice, we need to (i) construct the relevant causal graph and make sure the other endogenous variables ($\overrightarrow{W}$) can be controlled, and (ii) find an alternative fusion method ($\overrightarrow{x'}$) which is likely to avoid the original collision.

***Causal Relations Analysis.*** We construct a causal graph (Figure 8) specifying the relevant variables based on domain knowledge, the understanding of the source code of Openpilot and the carla simulator, and simulation runtime behavior across multiple runs. The exogenous variables include test design and the state of the system running simulation (e.g., real-time CPU workload, memory usage, etc.). Based on the understanding of ADAS scenario-based testing (see Section 6), test design influences the simulation result indirectly through determining scenario to test, simulator configurations, and Openpilot configurations (including the fusion method). Based on the understanding of the source code, simulator configurations can be further split into communication configurations and other configurations. Similarly, Openpilot configurations can be split into fusion method, communication configurations, and other configurations. The other exogenous variable system state indirectly influences the collision result via an endogenous variable communication latency. This is based on our observation that, in a system with limited CPU capacity available, the latency of the sensor information passed from the simulator to Openpilot can become very high and influences the collision result. Communication latency collectively represents the real-time latency of the communications between the simulator and Openpilot as well as among each of their sub-components, and thus captures the influence of the communication configurations of simulator and Openpilot, as well as the system state. We assume that all the variables directly influencing the occurrence of a collision have been included in the graph.

*Intervention for Eliminating Uncontrollable Influential Variable.* To check for causality, we need to be able to control the endogenous variables $\overrightarrow{W}$ and block any influence of the unobserved exogenous variables on the collision. With the default simulator and Openpilot communication configurations, communication latency (both between and within each of the simulator and Openpilot) influences the collision result and prevents a deterministic simulation replay. However, we cannot control the communication latency since one of its parents – the system state cannot be observed and controlled. To address this issue, we set the communication configurations of the simulator and the Openpilot to be deterministic and synchronous (see Appendix C in the extended version [44] for details). The communication latency then becomes zero thus avoiding the potential side effects [22]. Note such change is kept throughout the entire fuzzing process. We verify that no other uncontrollable influential variables on the collision results exist after this intervention in RQ1 by checking the reproducibility of the simulation results when using the same endogenous variables.

*Intervention for Cause Analysis.* During the fusion error analyzing step, we replace the initial fusion method ($\overrightarrow{x}$) with another fusion method ($\overrightarrow{x'}$) and check if a collision still happen. This step is regarded as an intervention on the fuzzing method after fuzzing.

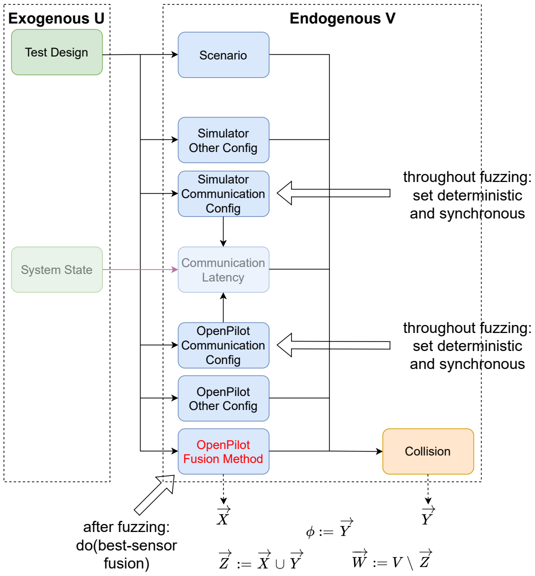

Exogenous U
Endogenous V
Test Design
Scenario
Simulator Other Config
Simulator Communication Config
throughout fuzzing: set deterministic and synchronous
System State
Communication Latency
OpenPilot Communication Config
throughout fuzzing: set deterministic and synchronous
OpenPilot Other Config
OpenPilot Fusion Method
Collision
after fuzzing: do(best-sensor fusion)
$\overrightarrow{X}$
$\overrightarrow{Y}$
$\phi := \overrightarrow{Y}$
$\overrightarrow{Z} := \overrightarrow{X} \cup \overrightarrow{Y}$
$\overrightarrow{W} := V \setminus \overrightarrow{Z}$

**Figure 8: Illustrating the causal graph with intervention.**

***Fusion Replacement Analysis.*** The next step is to efficiently find a fusion method $x'$ avoiding collision. The fusion method $x'$ should possess additional properties like having no extra knowledge and being functional. It should not have extra knowledge (e.g., the ground-truth of the locations of the NPC vehicles) beyond what it receives from the upstream sensor modules. Being functional means it should be good enough to enable the ego car to finish the original task. A counter-example is if the fusion method always false positively report the presence of a stationary NPC vehicle ahead and leads the ego car to stay stationary all the time.

To illustrate this, we define three different classes of fusion methods. Given everything else is kept the same, *collision fusion class* and *non-collision fusion class* consist of the fusion methods that lead to and avoid the collision, respectively. *No extra knowledge & functional class* consists of fusion methods which have no extra knowledge and are functional. If an failure is caused by the fusion method and can be fixed by changing it to a no extra knowledge and functional fusion method, there should be an intersection between non-collision fusion and no extra knowledge & functional fusion as shown in Figure 9(a) and Figure 9(c). Otherwise, there should be no intersection as shown Figure 9(b). The initial fusion method should fall into the intersection of the collision fusion class and the no extra knowledge & functional fusion class since a collision happens and it is reasonable to assume it (DEFAULT or MATHWORKS) has no extra knowledge and is functional.

In the current work, for each found collision, we only run one extra simulation to check if the fusion method is the cause. In particular, we set $\overrightarrow{x'}$ to *best-sensor fusion*. Note that this fusion method is an oracle fusion method since in reality we won't be able to know the ground-truth. However, it serves a good proxy. First, it uses no additional knowledge except for using the ground-truth to select the best sensor output. In reality, an ideal fusion method might potentially select the most reliable upstream sensor's prediction even without additional knowledge. Second, it is functional since it provides more accurate prediction than methods like DEFAULT and thus should be able to finish the original cruising task. Third, if *best-sensor fusion* cannot help to avoid a collision after the replacement, there is a high possibility that the collision is not due to the fusion method. The reason is that it already picks the best sensor prediction and thus does not make fusion fault, and it is reasonable to assume that the downstream modules perform better given its output compared with those less accurate outputs.

Thus, best-sensor fusion serves as a proxy to check if there is an intersection between no extra knowledge & functional fusion and non-collision fusion. If best-sensor fusion can help avoid the collision, the failure will be considered a fusion error. Otherwise, it will be discarded. There are three situations: (a) the failure is caused by the fusion method and the best-sensor fusion falls into non-collision fusion class(Figure 9a). (b) the failure is not caused by the fusion method and the best-sensor fusion does not fall into non-collision fusion class(Figure 9b). (c) the failure is caused by the fusion method and the best-sensor fusion does not fall into non-collision fusion class(Figure 9c). (a) and (b) are the true positive and true negative cases since the causation of the fusion method is consistent with the collision results of the best-sensor fusion method, while (c) is the false negative case. It also should be noted that there is no false positive case since if best-sensor fusion helps avoiding the collision, according to our reasoning earlier, the causation must hold. The implication is that a predicted fusion error is a failure caused by the fusion method but the reverse does not always hold.

# 5 RESULTS

To evaluate *FusED*, we explore the following research questions:

**RQ1: Evaluating Performance.** How effectively can *FusED* find fusion errors in comparison to baselines?

**RQ2: Case Study of Fusion Errors.** What are the representative causes of the fusion errors found?

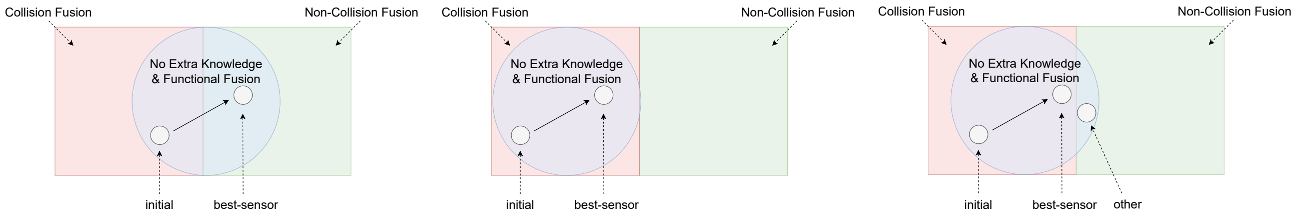
**RQ3: Evaluating Repair Impact.** How to improve MSF in OPENPILOT based on our observations on found fusion errors?

## 5.1 Experimental Design

**Environment.** We use CARLA 0.9.11 [16] as the simulator and OPENPILOT 0.8.5 as the ADAS [14]. The experiments run on a Ubuntu20.04 desktop with Intel i9-7940x, Nvidia 2080Ti, and 32GB memory.

**Studied Fusion Methods.** We apply *FusED* on DEFAULT and MATHWORKS introduced in Section 2.2.

**Driving Environments.** We utilize two driving environments named S1 and S2. S1 is a straight local road and S2 is a left curved highway road. Both S1 and S2 have 6 NPC vehicles. An illustration is shown in Figure 10 (not all NPC vehicles are shown). The maximum allowed speed of the auto-driving car is set to 45 miles/hr (≈ 20.1m/s) on the highway road (S2) and 35 miles/hr (≈ 15.6m/s) on the local road (S1). For vehicle types, S2 only considers cars and trucks while S1 additionally includes motorcycles and bicyclists. The search space for each vehicle consists of its type, its speed and



**(a) True Positive** **(b) True Negative** **(c) False Negative**

**Figure 9: An illustration of three situations of replacing the fusion method.**

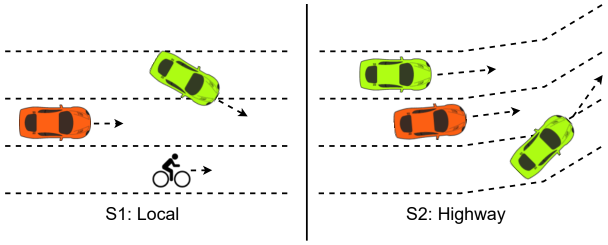


**Figure 10: An illustration of the two driving environments where the orange car represents the ego car.**

lane change decision (turn left/right, or stay in lane) at each time interval. Weather and lighting conditions are also considered. See Appendix E and Appendix F in the extended version [44] for the details of the driving environments and how they comply to the capability of OPENPILOT, respectively.

**Baselines and Metrics.** We use random search (RANDOM) and genetic algorithm without $\mathbf{F}_{\text{fusion}}$ in the fitness function (GA) as two baselines. We set the number of scenarios causing fusion errors and distinct fusion errors (Definition 3) as two evaluation metrics.

**Hyper-parameters.** We set the default values for $c_{\text{failure}}, c_d, c_{\text{fusion}}$ in the fitness function to −1, 1, −2. Since the crash-inducing property has two terms ($c_{\text{failure}}$ and $c_{\text{d}}$) while the fusion aspect has one ($c_{\text{fusion}}$), the default values balances the two's contribution. The sign for $c_{\text{d}}$ is positive since we want to minimize $F_{\text{d}}$ and the signs for the other two are negative since we want to maximize $F_{\text{failure}}$ and $F_{\text{fusion}}$. We set the pre-crash period's $m$ to 2.5 seconds because several states in US use 2.5s as the standard driver reaction time and studies have found the 95 percentile of perception-reaction time for human drivers is 2.5s [27]. Besides, in our context, when a fusion-induced collision happens, it is often caused by the fusion component's failure for about 2.5s before the collision as in Figure 4. We set $s$ and $l$ (defined in Section 3.4) to 30 and 10 such that each road interval is about 5m and each speed interval is about 4m/s. By default, we fuzz for 10 generations with 50 simulations per generation; each simulation runs at most 20 simulation seconds.

## 5.2 RQ1: Evaluating Performance

We compare GA-FUSION with the two baselines. Figure 11 shows the average number of fusion errors found by the three methods over three runs for each setting. On average, GA-FUSION has found 65%, 27%, 23%, and 44% more fusion errors than the best baseline method under each setting, respectively. Figure 12 shows the average number of distinct fusion errors (based on Definition 3) found by the three methods over three runs for each setting. GA-FUSION has also found 58%, 31%, 25%, and 37% more distinct fusion errors than the best baseline method, respectively. To test the significance of the results, we further conduct Wilcoxon rank-sum test [12] and Vargha-Delaney effect size test [8, 41] between the number of distinct fusion errors found by GA-FUSION and the best baseline under each setting. For each of the four settings, we have the p-value 0.05 and VD effect size interval (0.68, 1.32) at the 90% confidence interval, suggesting the difference is significant and the difference has medium effect size. These results show the effectiveness of *FusED* and superiority of GA-FUSION. The result also holds under different pre-crash period $m$ (see Appendix G in the extended version [44].

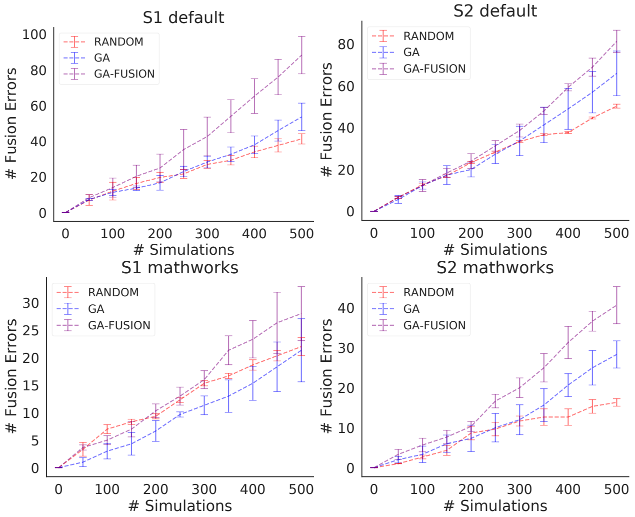

S1 default S2 default S1 mathworks S2 mathworks

**Figure 11: # fusion errors found over # simulations.**

The proposed GA-FUSION can efficiently find more fusion errors because: (1) $\mathbf{F}_{\text{d}}$ and $\mathbf{F}_{\text{failure}}$ can differentiate collisions (including both fusion errors and non-fusion errors) from no-collision, and (2) $\mathbf{F}_{\text{fusion}}$ can differentiate fusion errors from non-fusion errors. The first point is straightforward since when a collision happens, $\mathbf{F}_{\text{d}}$ is usually smaller and $\mathbf{F}_{\text{failure}}$ is 1. To show the second point, we plot the empirical cumulative density functions (ECDFs) of $\mathbf{F}_{\text{fusion}}$ for no-collision, non-fusion errors, and fusion errors, respectively, when running GA-FUSION under each setting.

As shown in Figure 13, on average, fusion errors tend to have larger $\mathbf{F}_{\text{fusion}}$ than non-fusion errors. We also apply Two-sample

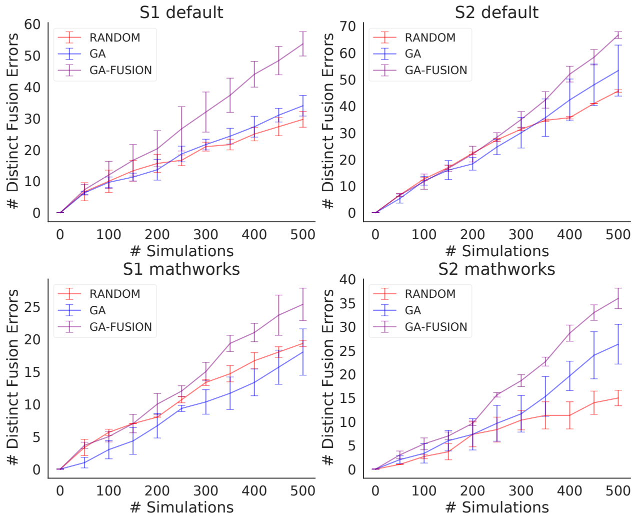
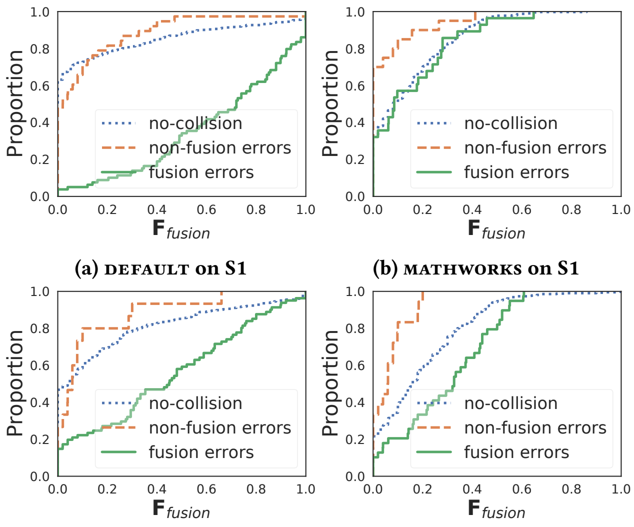


**Figure 12: # distinct fusion errors found over # simulations.**



(a) DEFAULT on S1

(b) MATHWORKS on S1

(c) DEFAULT on S2

(d) MATHWORKS on S2

**Figure 13: Empirical Cumulative Density Functions (ECDFs) of $\mathbf{F}_{\text{fusion}}$ for the three groups under the four settings. For each group, at a given x-axis value, the y-axis value is the proportion of scenarios in the group that have $\mathbf{F}_{\text{fusion}}$ less than or equal to the given x-axis value. The plots show that a larger portion of fusion errors have larger $\mathbf{F}_{\text{fusion}}$ than non-fusion errors so $\mathbf{F}_{\text{fusion}}$ can help to differentiate the two.**

Kolmogorov–Smirnov test [38] on the ECDFs of $\mathbf{F}_{\text{fusion}}$ for fusion errors and non-fusion errors. The test statistic versus the corresponding 0.05 significance threshold for each setting are 0.76>0.27, 0.42>0.39, 0.58>0.38, and 0.67>0.39, respectively [26, 33], showing the fusion errors and non-fusion errors differ at the 0.05 significance level under each setting.

Note that in Figure 13(b), fusion errors have similar $\mathbf{F}_{\text{fusion}}$ as no-collision. This can happen when fusion method is faulty but that may not result in collision. Typical examples are scenarios in which a leading vehicle's relative longitudinal distance is wrongly predicted but no collision happens since it is very far away from OPENPILOT. However, these fusion errors are still more likely to be selected at the selection stage since fusion errors involve the occurrence of collisions which result in smaller $\mathbf{F}_{\text{d}}$ and larger $\mathbf{F}_{\text{failure}}$ than no-collision.

**Sanity Check of the Causal Graph.** In order to make sure the causal graph (Figure 8) includes all the influential variables on the collision result, from the scenarios we have run during the fuzzing process, we randomly selected 100 collision scenarios and 100 no-collision scenarios, and run them again with every controllable endogenous variable kept the same. All the repeated runs reproduce the collision/no-collision results. This implies no influential variables are likely to be omitted, since if such variables exist, repeated runs with the same endogenous variables should lead to different simulation results.

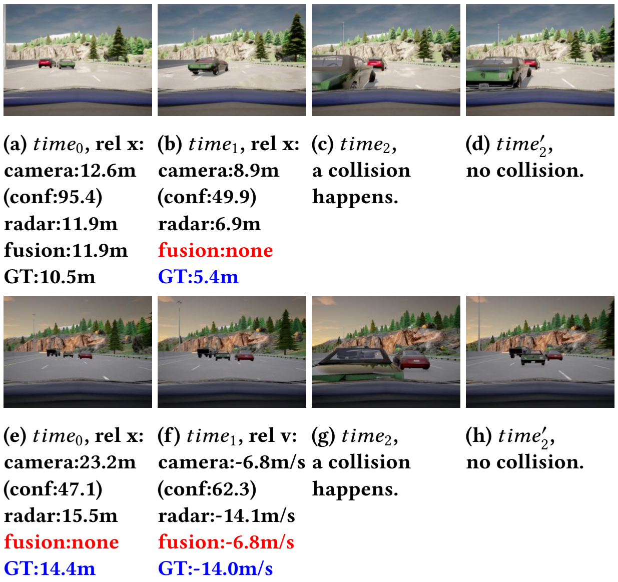
> **Result 1:** *Under each of the four settings, at the 0.05 significance level, FusED finds more distinct fusion errors (as well as fusion errors) than the best baseline method. The difference has a medium effect size at 90% confidence interval.*

## 5.3 RQ2: Case Study of Fusion Errors

In this subsection, we show three representative fusion errors found by *FusED* and analyze their root causes.

**Case1: Incorrect camera lead dominates accurate radar lead.** The first row of Figure 14 shows a failure due to ② in Figure 3a. In Figure 14a, both camera and radar give accurate prediction of the leading green car at $time_0$. At $time_1$ in Figure 14b, the green car tries to change lane, collides with a red car, and blocks the road. The camera model predicts that the green car with a low confidence (49.9%) and the fusion component thus misses all leading vehicles due to ② in Figure 3a. The ego car keeps driving until hitting the green car at $time_2$ in Figure 14c. If the radar data is used instead from $time_0$, however, the collision can be avoided, as shown in Figure 14d. Going back to Figure 3a, the root cause is that OPENPILOT prioritizes the camera prediction and it ignores any leading vehicles if the camera prediction confidence is below 50%. As a result, despite the accurate information predicted by the radar, OPENPILOT still causes the collision.

**Case2: Inaccurate radar lead selected due to mismatch between radar and camera.** The second row of Figure 14 shows another failure caused by both ② and ③ in Figure 3a. At $time_0$ of Figure 14e, camera overestimates the longitudinal distance to the leading green car. At $time_1$ of Figure 14f, though one radar data (not shown) is close to the correct information of the green car, the radar data of the Cybertruck on the left lane matches the camera's prediction. Thus, the Cybertruck lead data is selected regarding ③ in Figure 3a. Consequently, although the ego car slows down, the process takes longer time than if it selects the green car radar data. This finally results in the collision at $time_2$ in Figure 14g. If the green car radar lead is used from $time_0$, the ego car would slow down quickly and thus not hit the green car at $time_2'$ in Figure 14h. This failure also correlates to camera dominance but it additionally involves mismatching in ③ of Figure 3a.



(a) $time_0$, rel x: camera:12.6m (conf:95.4) radar:11.9m fusion:11.9m GT:10.5m

(b) $time_1$, rel x: camera:8.9m (conf:49.9) radar:6.9m fusion:none GT:5.4m

(c) $time_2$, a collision happens.

(d) $time_2'$, no collision.

(e) $time_0$, rel x: camera:23.2m (conf:47.1) radar:15.5m fusion:none GT:14.4m

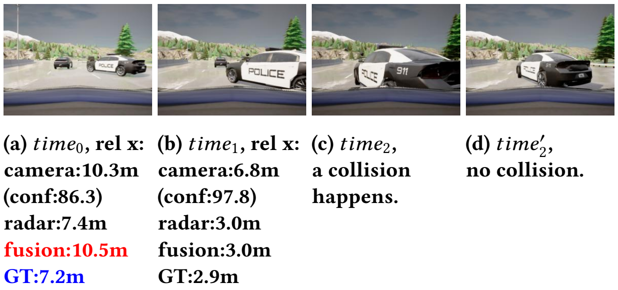
(f) $time_1$, rel v: camera:-6.8m/s (conf:62.3) radar:-14.1m/s fusion:-6.8m/s GT:-14.0m/s

(g) $time_2$, a collision happens.

(h) $time_2'$, no collision.

**Figure 14: Two found fusion errors for DEFAULT. *rel v* represents relative speed to the leading NPC vehicle.**

**Case3: Discarding correct lead due to a faulty selection method.** Figure 15 shows an example when MATHWORKS fails due to ③ in Figure 3b. At $time_0$ in Figure 15a, a police car on the right lane is cutting in. While radar gives a very accurate prediction of the red car, the radar prediction is not used since ③ in Figure 3b only selects among the predicted leads within the current lane. Consequently, a camera predicted lead is used, which overestimates the relative longitudinal distance. At $time_1$ in Figure 15b, the correct radar prediction is used but it is too late for the ego car to slow down, causing the collision at $time_2$ in Figure 15c. If the best predicted lead (i.e. the one from the radar data) is used starting at $time_0$, the collision would disappear at the time $time_2'$ of Figure 15d.



(a) $time_0$, rel x: camera:10.3m (conf:86.3) radar:7.4m fusion:10.5m GT:7.2m

(b) $time_1$, rel x: camera:6.8m (conf:97.8) radar:3.0m fusion:3.0m GT:2.9m

(c) $time_2$, a collision happens.

(d) $time_2'$, no collision.

**Figure 15: A found fusion error for MATHWORKS.**

**Result 2:** *The representative fusion errors found by FusED for the two fusion methods are due to the dominance of camera over radar, their mismatch, or the faulty prediction selection method.*

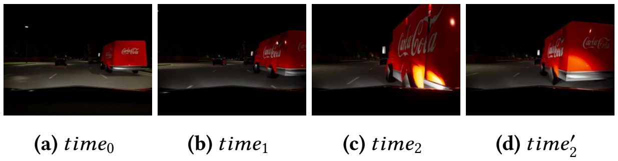
## 5.4 RQ3: Evaluating Repair Impact

To avoid the fusion errors, one obvious alternative to the studied fusion methods seems to simply let the radar predictions dominate the camera predictions as the examples shown in Section 5.3 are mainly caused by the dominance of unreliable camera prediction. Such design, however, suffers from how to choose the fusion leads from all radar predicted leads. If the fusion method simply chooses the closest radar lead and that lead corresponds to an NPC vehicle on a neighboring lane, the ego car may never pass the NPC vehicle longitudinally even when the NPC vehicle drives at a low speed.

**Table 1: # avoided / # distinct fusion errors.**

| S1 DEFAULT | S1 MATHWORKS | S2 DEFAULT | S2 MATHWORKS |
|---|---|---|---|
| 43/52 | 14/26 | 67/78 | 13/26 |

Based on our observation and analysis of the above fusion errors, we suggest two improvements to enhance the fusion methods. First, radar predictions should be integrated rather than dominated by camera predictions (Cases 1-2). Second, vehicles intending to cut in should be tracked and considered (Case 3). MATHWORKS already addresses the first aspect and thus has less fusion errors found. Regarding the second one, for each tracked object by radar, we store their latitudinal positions at each time step. At next time step, if a vehicle's relative latitudinal position gets closer to the ego car, it will be included in the candidate pool for the leading vehicle rather than discarded. We call this new fusion method MATHWORKS+.



(a) $time_0$ (b) $time_1$ (c) $time_2$ (d) $time_2'$

**Figure 16: An fusion error avoided by MATHWORKS+.**

We evaluate MATHWORKS+ via replacing the original fusion method with it during the pre-crash window on the previously found fusion errors. As shown in Table 1, at least 50% of found fusion errors can be avoided. Figure 16 shows a fusion error found on MATHWORKS but avoided by MATHWORKS+. At $time_0$ (Figure 16a), the ego car and a red truck drive on different lanes. At $time_1$, MATHWORKS does not consider the truck since it just starts to invade into the current lane (the truck's radar lead is discarded at ③ in Figure 3b). When the truck fully drives into the current lane, it is too late for the ego car to avoid the collision at $time_2$ (Figure 16c). If MATHWORKS+ is used since Figure 16a, the truck would be considered a leading vehicle at Figure 16b and the collision would be avoided at $time_2'$ (Figure 16d). These results demonstrate the improvement of MATHWORKS+. Further, it implies that with a good fusion method, many fusion errors can be avoided without modifying the sensors or the processing units.

**Result 3:** *Based on the observations of the found fusion errors, we adjust the fusion method we study and enable it to avoid more than 50% of the initial fusion errors.*

## 6 RELATED WORK

**Search Based Software Engineering(SBSE).** SBSE formulates a software engineering problem into a search problem and applies search-based metaheuristic optimization techniques [19, 20, 35]. In our context, we apply evolutionary algorithm to search for test cases (scenarios) which can cause ego car's fusion errors.

**Scenario-Based Testing(SBT).** In order to identify the errors of a driving automation system, comprehensive tests are being conducted by autonomous driving companies. The public road testing approach is the closest to a system's use case, but it is incredibly costly. It has been shown that more than 11 billion miles are required to have a 95% confidence that a system is 20% safer than an average human driver [25]. Most of these miles, however, usually do not pose threats to the system under test and are thus not efficient, if not wasted. To focus on challenging test cases, SBT techniques have been developed where a system is tested in difficult scenarios designed by experts or found by algorithms. Besides, since many dangerous cases (e.g., a pedestrian crossing a street close to the ego car) cannot be tested in the real world, such tests are usually conducted in a high-fidelity simulator [46]. Existing works usually treat the system under test as a black-box and search for hard scenarios to trigger ego car's potential failure. Search methods leveraging evolutionary algorithms [2, 10, 28, 45], reinforcement learning [13], bayesian optimization [5], and topic modeling[15] have been used.

The failures found can have different causes like the failure of the sensors, the planning module, or the modules' interactions. However, most existing works either ignore the root cause analysis or merely analyze causes for general failures. In contrast, we focus on revealing failures causally induced by the fusion component. Abdessalem et al. [3] study the failure of an ADAS's integration component which integrates the decisions of different functionalities (e.g., AEB and ACC). In contrast, we focus on the fusion component which integrates the data from multiple sensors. Besides, the fusion component can either be rule-based (e.g., DEFAULT) or algorithm-based (e.g., MATHWORKS) while the integration component studied in [3] is only rule-based.

**Adversarial Attacks on Fusion.** Some recent works study how to attack the fusion component of an automated system and thus fails the system [11, 37, 40]. In particular, Cao et al. [11] and Tu et al. [40] study how to construct adversarial objects that can fool both camera and Lidar at the same time and thus lead the MSF to fail. Shen et al. [37] study how to send spoofing GPS signals to confuse a MSF on GPS and Lidar. In contrast to creating artificial adversarial objects or sending adversarial signals, we focus on finding scenarios under which the faults of MSF lead to critical accidents without the presence of any malicious attacker.

## 7 THREATS TO VALIDITY

There remains a gap between testing in real-world and testing in a simulator. However, road testing is overly expensive and not flexible. Besides, a simulation environment allows us to run counterfactual simulations easily and attribute an failure to the fusion method used. Consequently, we focus on testing in the simulation environment.

The causal graph (Figure 8) constructed may not capture all the influential variables. To mitigate this threat, in RQ1, we run sanity check of the causal graph by checking the reproducibility of the simulation results when using the same endogenous variables.

Since we change the communication within Openpilot and that between Openpilot and carla to be synchronous and deterministic (see Appendix C in the extended version [44]), the behavior of Openpilot can be different from the original Openpilot. However, this change should not influence the found fusion errors since Openpilot should perform better when it receives the latest sensor data rather than the delayed ones.

As in [2, 31], we evaluate the proposed method using the number of found scenarios leading to fusion errors. However, we have observed that this metric might double-count similar fusion errors. To mitigate this threat, we additionally use another counting metric based on the ego car's trajectory (Section 3.4).

Besides, similar to previous works[2, 28], we evaluate the studied ADAS in two driving environments. Since ADAS only performs the task of lane following, the complexity of its applicable environments is limited and thus mostly covered in the two environments.

Another threat is that the hyper-parameters are not fine-tuned. However, even with the current parameters, the proposed method already outperforms the baselines. We believe that the performance of the proposed method can be improved by fine-tuning.

Furthermore, we only test MATHWORKS+ on limited detected fusion errors on Openpilot. There might be corner cases that are not covered. Since we focus on fusion errors finding rather than fixing, we leave a comprehensive study for future work.

Finally, the current fusion objective only applies to HLF and is only tested on two popular fusion methods in Openpilot. Conceptually, the proposed method can generalize to the fusion component in ADS like Apollo[9] and Autoware[39] which use HLF components. We plan to study other types of fusion methods like MLF and LLF, as well as MSF in ADS in future work.

## 8 CONCLUSION

In this work, we formally define, expose, and analyze the root causes of fusion errors on two widely used MSF methods in a commercial ADAS. To the best of our knowledge, our work is the first study on finding and analyzing failures causally induced by MSF in an end-to-end system. We propose a grey-box fuzzing framework, *FusED*, that effectively detects fusion errors. Lastly, based on the analysis of the found fusion errors, we provide several learned suggestions on how to improve the studied fusion methods.

## DATA AVAILABILITY STATEMENT

The artifact consists of the source code of *FusED* has been made publicly available [17].

## ACKNOWLEDGEMENT

The majority of work was done during the internship of Ziyuan Zhong at Baidu Security X-Lab. The work is also supported in part by NSF CCF-1845893, CCF-2107405, and IIS-2221943. We want to thank colleagues from Baidu Security X-Lab and Chengzhi Mao from Columbia University for valuable discussions.

## REFERENCES

[1] 2022. Tesla Deaths. https://www.tesladeaths.com/.

[2] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 1016–1026. https://doi.org/10.1145/3180155.3180160

[3] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-Objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/3238147.3238192

[4] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2020. Automated Repair of Feature Interaction Failures in Automated Driving Systems. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 88–100. https://doi.org/10.1145/3395363.3397386

[5] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek. 2019. Generating Adversarial Driving Scenarios in High-Fidelity Simulators. In *2019 International Conference on Robotics and Automation (ICRA)*. 8271–8277.

[6] National Highway Traffic Safety Administration. 2020. Common Driver Assistance Technologies. (2020). https://www.nhtsa.gov/equipment/driver-assistance-technologies

[7] Ram Agrawal, Kalyanmoy Deb, and Ram Agrawal. 2000. Simulated Binary Crossover for Continuous Search Space. *Complex Systems* 9 (06 2000).

[8] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250. https://doi.org/10.1002/stvr.1486

[9] BaiduApolloTeam. 2021. Apollo: Open Source Autonomous Driving. https://github.com/ApolloAuto/apollo. Accessed: 2019-02-11.

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

[11] Yulong Cao, Ningfei Wang, Chaowei Xiao, Dawei Yang, Jin Fang, Ruigang Yang, Q. Chen, Mingyan D. Liu, and Bo Li. 2021. Invisible for both Camera and LiDAR: Security of Multi-Sensor Fusion based Perception in Autonomous Driving Under Physical-World Attacks. *ArXiv* abs/2106.09249 (2021).

[12] J. Anthony Capon. 1991. Elementary Statistics for the Social Sciences: Study Guide. (1991).

[13] Baiming Chen and Liang Li. 2020. Adversarial Evaluation of Autonomous Vehicles in Lane-Change Scenarios.

[14] CommaAI. 2021. Openpilot. https://github.com/commaai/openpilot

[15] Wenhao Ding, Baiming Chen, Minjun Xu, and Ding Zhao. 2020. Learning to Collide: An Adaptive Safety-Critical Scenarios Generating Method. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Virtual). https://arxiv.org/abs/2003.01197

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator *(Proceedings of Machine Learning Research, Vol. 78)*, Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (Eds.). PMLR, 1–16. http://proceedings.mlr.press/v78/dosovitskiy17a.html

[17] Elysium. 2022. *AIasd/FusED: Initial Release*. https://doi.org/10.5281/zenodo.6516024

[18] Joseph Y. Halpern. 2015. A Modification of the Halpern-Pearl Definition of Causality. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) *(IJCAI'15)*. AAAI Press, 3022–3033.

[19] Mark Harman and Bryan F Jones. 2001. Search-based software engineering. *Information and Software Technology* 43, 14 (2001), 833–839. https://doi.org/10.1016/S0950-5849(01)00189-6

[20] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. 2012. *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–59. https://doi.org/10.1007/978-3-642-25231-0_1

[21] Zhisheng Hu, Shengjian Guo, Zhenyu Zhong, and Kang Li. 2021. Coverage-based Scene Fuzzing for Virtual Autonomous Driving Testing. *arxiv* (2021).

[22] Zhisheng Hu, Shengjian Guo, Zhenyu Zhong, and Kang Li. 2021. Disclosing the Fragility Problem of Virtual Safety Testing for Autonomous Driving Systems. In *IEEE International Symposium on Software Reliability Engineering, ISSRE 2021 - Workshops, Wuhan, China, October 25-28, 2021*. IEEE, 387–392. https://doi.org/10.1109/ISSREW53611.2021.00106

[23] Consumer Reports Data Intelligence. 2020. Active Driving Assistance Systems:Test Results and Design Recommendations. https://data.consumerreports.org/wp-content/uploads/2020/11/consumer-reports-active-driving-assistance-systems-november-16-2020.pdf

[24] SAE International. 2021. J3016 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. (2021).

[25] Nidhi Kalra and Susan M. Paddock. 2016. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation. http://www.jstor.org/stable/10.7249/j.ctt1btc0xw

[26] Donald E. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA.

[27] Robert Layton and Karen Dixon. 2012. Stopping Sight Distance. (2012).

[28] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012

[29] Yue Li, Devesh K. Jha, Asok Ray, and Thomas A. Wettergren. 2015. Feature level sensor fusion for target detection in dynamic environments. In *2015 American Control Conference (ACC)*. 2433–2438. https://doi.org/10.1109/ACC.2015.7171097

[30] Yuzhe Ma, Jon A. Sharp, Ruizhe Wang, Earlence Fernandes, and Xiaojin Zhu. 2021. Sequential Attacks on Kalman Filter-based Forward Collision Warning Systems. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 8865–8873. https://ojs.aaai.org/index.php/AAAI/article/view/17073

[31] R Majumdar, A Mathur, M Pirron, L Stegner, and D. Zufferey. 2021. Paracosm: A Test Framework for Autonomous Driving Simulations. *Fundamental Approaches to Software Engineering* (2021).

[32] V. Manes, H. Han, C. Han, s. cha, M. Egele, E. J. Schwartz, and M. Woo. 2019. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering* 01 (oct 2019), 1–1. https://doi.org/10.1109/TSE.2019.2946563

[33] Frank J Massey. 1952. Distribution table for the deviation between two sample cumulatives. *The annals of mathematical statistics* 23, 3 (1952), 435–441.

[34] Mathwork. 2021. Forward Collision Warning Using Sensor Fusion.

[35] Phil McMinn. 2011. Search-Based Software Testing: Past, Present and Future. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. 153–163. https://doi.org/10.1109/ICSTW.2011.100

[36] Judea Pearl. 1998. Graphical models for probabilistic and causal reasoning. *Quantified representation of uncertainty and imprecision* (1998), 367–389.

[37] Junjie Shen, Jun Won, Zeyuan Chen, and Qi Alfred Chen. 2020. Drift with Devil: Security of Multi-Sensor Fusion based Localization in High-Level Autonomous Driving under GPS Spoofing (Extended Version). In *USENIX Security Symposium 2020*.

[38] Nikolai V Smirnov. 1939. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bull. Math. Univ. Moscou* 2, 2 (1939), 3–14.

[39] TheAutowareFoundation. 2016. Autoware: Open-source software for urban autonomous driving. https://github.com/CPFL/Autoware

[40] James Tu, Huichen Li, Xinchen Yan, Mengye Ren, Yun Chen, Ming Liang, Eilyan Bitar, Ersin Yumer, and Raquel Urtasun. 2021. Exploring Adversarial Robustness of Multi-Sensor Perception Systems in Self Driving. (01 2021).

[41] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. https://doi.org/10.3102/10769986025002101 arXiv:https://doi.org/10.3102/10769986025002101

[42] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. 2021. Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. *Sensors* 21, 6 (2021). https://doi.org/10.3390/s21062140

[43] Jin Hyeok Yoo, Yecheol Kim, Jisong Kim, and Jun Won Choi. 2020. 3D-CVF: Generating Joint Camera and LiDAR Features Using Cross-view Spatial Feature Fusion for 3D Object Detection. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 720–736.

[44] Ziyuan Zhong, Zhisheng Hu, Shengjian Guo, Xinyang Zhang, Zhenyu Zhong, and Baishakhi Ray. 2021. Detecting Multi-Sensor Fusion Errors in Advanced Driver-Assistance Systems. https://doi.org/10.48550/ARXIV.2109.06404

[45] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2021. Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles. *arXiv preprint arXiv:2109.06126* (2021).

[46] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vânia de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation. *CoRR* abs/2112.00964 (2021). arXiv:2112.00964 https://arxiv.org/abs/2112.00964