Detecting Safety Problems of Multi-Sensor Fusion in Autonomous Driving

Ziyuan Zhong\textsuperscript{1}, Zhisheng Hu\textsuperscript{2}, Shengjian Guo\textsuperscript{2}, Xinyang Zhang\textsuperscript{2}, Zhenyu Zhong\textsuperscript{2}, Baishakhi Ray\textsuperscript{1}

\textsuperscript{1} Columbia University
\textsuperscript{2} Baidu Security

ziyuan.zhong@columbia.edu, zhishenghu@baidu.com, sguo@baidu.com, xinyangzhang@baidu.com, edwardzhong@baidu.com, rayb@cs.columbia.edu

Abstract

Autonomous driving (AD) systems have been thriving in recent years. In general, they receive sensor data, compute driving decisions, and output control signals to the vehicles. To smooth out the uncertainties brought by sensor inputs, AD systems usually leverage multi-sensor fusion (MSF) to fuse the sensor inputs 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. As a result, critical consequences might happen unexpectedly. In this work, we observed that the popular MSF methods in an industry-grade Advanced Driver-Assistance System (ADAS) can mislead the car control and result in serious safety hazards. Misbehavior can happen regardless of the used fusion methods and the accurate data from at least one sensor. To attribute the safety hazards to a MSF method, we formally define the fusion errors and propose a way to distinguish safety violations causally induced by such errors. Further, we develop a novel evolutionary-based domain-specific search framework, FusionFuzz, for the efficient detection of fusion errors. We evaluate our framework on two widely used MSF methods. Experimental results show that FusionFuzz identifies more than 150 fusion errors. Finally, we provide several suggestions to improve the MSF methods under study.

1 Introduction

A typical Autonomous Driving (AD) system, as shown in Figure \ref{fig:architecture}, takes inputs from a set of sensors (e.g. camera, LiDAR, radar, GPS, 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. It is believed that individual sensor data could be unreliable under various extreme environments. For example, a camera can fail miserably in a dark environment in which a LiDAR can function properly. A LiDAR can be highly inaccurate on a cloudy day where a radar can still provide precise measurements. To enable an AD 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 a sensor fails, MSF can still work with other sensors to provide reliable information for the downstream modules and enable the AD system operate normally.

However, MSF hardly knows which sensor provides the most accurate information. Thus, it neither thoroughly eliminates the uncertainty nor always weighs more on the right sensor data. This inherent flaw may introduce safety risks to the AD systems, especially the widely-used Advanced Driver-Assistance System (ADAS), a widely-used Level2 AD. In the study of a popular commercial ADAS named PENPILOT, we found that its default rule-based MSF occasionally produces inaccurate output despite an accurate input exists, and leads the entire system to critical accidents. Similarly, running PENPILOT with an alternative Kalman-filter based MSF also suffers from certain fusion-inducing failures. To improve the MSF reliability and assure the entire ADAS safety in most environments, understanding and identifying such hidden risks are essential, whereas we need to address two primary challenges.

Challenges. First, it is hard to perform causality forensics between safety violations and the fusion method. The fusion inputs can be highly non-deterministic and the fusion output might not instantly trigger a safety violation because it has to undergo subsequent planning and control modules to take effect. Thus, a critical violation may occur after the accumulation of many cycles of system executions. Second, existing fusion methods can function properly most of the time so their failure cases are extremely sparse among all the driving situations. Given testing an AD system is costly, it is non-trivial to identify the failure cases within limited time budget.

Our Approach. For the first challenge, we study if an error still happens after we replace the original fusion method with a new oracle-like fusion method in an otherwise identical setting. In the counterfactual theories of causation, an event A causes an event B, if and only if, if A were not to occur B would not occur. Therefore, after the replacement,
if the error no longer happens, the original fusion method would be considered the cause of the error. We call such error a fusion error. To ensure elements other than the fusion method being identical in both runs, we adopt a high-fidelity simulator CARLA (Dosovitskiy et al., 2017) to construct a stable virtual safety testing environment.

We next address the challenge on the difficulty of finding the failure cases of the fusion methods. In the AD testing domain, recent works leverage fuzz testing (fuzzing) to generate simulator input scenarios to run a AD system and search for failure-inducing scenes (Abdessalem et al., 2018a; Li et al., 2020; Hu et al., 2021; Zhong, Kaiser, and Ray, 2021). However, these methods treat the target system as a black-box and ignore the attainable runtime information of the AD system. Inspired by the grey-box fuzzing in software fuzzing literature (Manes et al., 2019), we propose an evolutionary algorithm-based fuzzing framework FusionFuzz that utilizes the input and output information of the fusion component of an AD system. In particular, we propose a novel objective function that simultaneously maximizes the deviation between the fusion output and the ground-truth and minimizes the deviation between a fusion input and the ground-truth. Intuitively, when both objectives are met, the fusion method must have made a bad decision since it fails to utilize an accurate input to generate a reasonable output. If the simulation then witnesses an error of the auto-driving car, the fusion output is the likely cause. Consequently, FusionFuzz can greatly ease the efforts in detecting fusion errors.

To the best of our knowledge, our technique is the first fuzzing method targeting the ADAS fusion component. In summary, we make the following contributions:

- We define and evaluate fusion errors in an industry-grade ADAS under a non-adversarial setting.
- We pinpoint the uncertainty introduced by the fusion component as the root cause of collisions and disclose safety issues of two popular fusion methods.
- We develop FusionFuzz, a novel grey-box fuzzing technique for efficiently revealing fusion errors in ADAS.
- We propose suggestions to mitigate fusion errors and effectively reduce fusion errors in a preliminary study.

## 2 Background

### 2.1 Simulation Testing for AD

Simulation testing for AD is widely used to complement real-world road testing. These works usually treat the AD system under test as a black-box and search for hard scenarios to check if the AV would collide or violate traffic rules (O’Kelly et al., 2018; Wheeler and Kochenderfer, 2019; Abeyesinghoonawardena, Shkurtt, and Dudur, 2019; Ding et al., 2020; Kuuti, Fallah, and Bowden, 2020; Chen and Li, 2020; Koren et al., 2018; Wenhao et al., 2020; Abdessalem et al., 2018a, b; Ben Abdessalem et al., 2016; Li et al., 2020; Hu et al., 2021; Zhong, Kaiser, and Ray, 2021). However, previous works either ignore the root cause of collisions and merely analyze reasons for general errors. In contrast, we focus on revealing errors causally induced by the fusion component.

### 2.2 Fusion in Autonomous Driving

#### Fusion Method

Most industry-grade AD systems leverage MSF to avoid potential accidents caused by the failure of a single sensor (BaiduApolloTeam, 2021; TheAutoWearFoundation, 2016; CommaAI, 2021). MSF often works with Camera and Radar, Camera and LiDAR, or the combination of Camera, Radar and LiDAR (Yeong et al., 2021). Fusion methods can be categorized into high-level, middle-level, and low-level methods. High-level fusion is particularly widely used because of its simplicity. A method in this category usually takes in high-level object attributes of the environment (e.g. the relative positions of nearby vehicles) processed from raw sensor data and outputs aggregate object attributes to its downstream components. In this work, we conduct a CARLA simulator-based case study on an industry-grade ADAS, OPENPILOT, which uses a high-level fusion method for camera and radar. We find that its fusion flow design is potentially problematic and fusion errors can be triggered under certain inputs. We further study a popular Kalman-Filter based fusion method (Mathwork, 2021; Ma et al., 2021), and use it to replace OPENPILOT’s default fusion method to evaluate it in the context of an entire system.

#### Fusion Attacks

Cao et al., 2021; Tu et al., 2021 create adversarial objects for both Camera and LiDAR to make a MSF fail. Shen et al., 2020 send spoofing GPS signals to confuse a MSF on GPS and LiDAR. In comparison, we focus on finding scenarios under which the failure of MSF leads to critical accidents without adversarial attacks.

## 3 Fusion in OPENPILOT

In this section, we first introduce the role of fusion in OPENPILOT, a commercial ADAS that provides the functionality of adaptive cruise control and lane centering. Next, we give a motivating example of fusion misbehavior inducing accident of OPENPILOT. We then review the fusion logic in OPENPILOT (denoted as ORIGINAL) to explain the accident. We further introduce a Kalman-filter based fusion method, denoted as MATHWORK, as an additional studied MSF.

### 3.1 Overview

![Figure 2: The role of fusion in OPENPILOT.](image-url)

Shown in Figure 2 the fusion component in OPENPILOT 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, relative longitudinal and lateral distances to the leading vehicle, and the confidence of this prediction (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. Note that the latitudinal control component only relies on camera data so we do not consider accidents due to the auto-driving car driving out of the lane.

3.2 A Collision Example

Figure 3: A collision example. rel: x: relative longitudinal distance to the leading vehicle. c: the best camera predicted lead with confidence. r: the best radar predicted lead. f: the fusion prediction. gt: the ground-truth value.

Figure 4 shows an example where the ego car collides with a bicyclist cutting in. At $t_0$ (Figure 4a), no leading vehicle exists. At $t_1$ (Figure 4b), the bicyclist on the right trying to cut in. While radar predicted lead is close to the ground-truth, 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 $t_2$ (Figure 4c). This example shows an accident caused by a wrong result from the fusion component. But how could the problem happen?

3.3 The Internal Logic of Fusion Methods

**ORIGINAL: Heuristic Rule-based Fusion.** Figure 4a shows the logic flow of the fusion component in OPENPILOT. It first checks if the auto-driving car is at a low speed 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 0.5 (②). If not, leading vehicles will be considered non-existent. Otherwise, it checks if any radar leads match the camera leads. If so, the best-matching radar leads are returned. Otherwise, the camera leads are returned. This logic explains the failure in Figure 5. In particular, due to ② in Figure 5a, no leading vehicle is considered existent at $t_1$ so the ego car accelerates until hitting the bicyclist.

**MATHWORK: Kalman-Filter Based Fusion.** Note that various fusion methods are available in addition to the ORIGINAL. Figure 4b shows the logic of another popular fusion method from Mathwork. It starts with the camera-predicted lane to filter out cluttered (i.e. stationary outside the auto-driving 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. In this work, we also involve MATHWORK fusion method for the evaluation.

4 The Fusion Error

In this section, we provide a formal definition of the fusion errors and design a way to distinguish them.

4.1 The Definition

We deem that a fusion error should have three properties as to be critical, avoidable, and fusion-induced.

**Critical:** a simulation should witness an accident of the ego car. Since only the longitudinal control module in OPENPILOT relies on fusion, we focus on the collision accidents.

**Avoidable:** the happened accident could be avoidable. We focus on the cases where the ego car has enough time to stop before the collision. Besides, the collision must happen within the field of view of the ego car so OPENPILOT’s camera is able to see the NPC (Non-player character) vehicle.

**Fusion-induced:** the critical and avoidable simulation collision should be causally induced by the misbehavior of the fusion component.

Next, we define several terminologies. As depicted in Figure 5, a pre-crash period consists of the $m$ seconds before an accident. Given all the predicted leads, the best-sensor fusion is a MSF that always selects the right one that is closest to the ground-truth lead. In the counterfactual theories of causation [Menzies and Beebee 2020], causal dependence is:

"Where $c$ and $e$ are two distinct actual events, $e$ causally depends on $c$ if and only if, if $c$ were not to occur $e$ would not occur."

Inspired by it, we make the formal definition:

**Definition 1.** An error of ego car is considered a fusion error if the error would not occur once the best-sensor fusion is used during the pre-crash period in a counter-factual world.

Figure 5 illustrates this definition. At ①, a simulation starts and OPENPILOT is engaged. The simulation enters the pre-crash period at ② and finally a collision happens at ③. If the best-sensor fusion is used from ②, in the counter-factual
Figure 5: An illustration of pre-crash period and fusion error.

world, no collision happens at 3. Note that the condition in the definition is sufficient but not necessary. The reason is that an ideal fusion method can even outperform the best-sensor fusion since it also interpolates among the leads data.

4.2 Error Counting

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 [R et al. 2021, Abdessalem et al. 2018a] 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 [Li et al. 2020]. Such way is subjective and time-consuming when the number of errors grows up.

Inspired by the location trajectory coverage metric [Hu et al. 2021], we consider the ego car’s state (in particular, position 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 $1 \times s \times l$. During the simulation, the ego car’s real-time location and speed are recorded at 2Hz. The recorded location-speed data points are then mapped to their corresponding “cells” on the coverage plane. Given all the data points mapped into the cells having the same road interval, their average speed is taken, the corresponding speed-road cell is considered “covered”, and the corresponding field on the coverage plane is assigned 1. Note a simulation’s final trajectory representation can have at most s non-zero fields although it has $s \times l$ fields in total.

We denote the trajectory vector associated with the simulation run for a specification $x$ to be $R(x)$ and define:

**Definition 2.** Two fusion errors for the simulations runs on specifications $x_1$ and $x_2$ are considered distinct if $| |R(x_1) - R(x_2)||_0 > 0$.

To demonstrate this error counting approach, we show two fusion errors with different trajectories in Figure 6. In both Figure 6a 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 in 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 2.

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.

5 FusionFuzz

In this section, we introduce FusionFuzz, our automated fuzzing framework for fusion errors detection.

5.1 The Overall Flow

Figure 7 shows the overall flow of FusionFuzz. It consists of two major components: the fuzzing engine (fuzzer) and the simulation. The fuzzer proceeds for predefined rounds of generations. At each generation, it feeds generated scenarios (also called seeds) into the simulation. In simulation, at each time step, the CARLA simulator supplies the sensor data of the current scene to PENPILOT. After PENPILOT sends back its control commands, the scene in CARLA updates. Once the simulations finish, any found fusion errors are reported while the seeds along with their objective values in the simulations are returned as feedback to the fuzzer. The fuzzer then leverages the feedback to generate new seeds in the execution of next generation.

5.2 Search Space

We utilize two driving environments named S1 and S2 in our study. S1 is a straight local road and S2 is a left curved highway road. Both S1 and S2 have 6 NPC vehicles running

![Figure 7: The overall workflow of FusionFuzz.](Image 340x556 to 411x609)

![Figure 3: Examples of two fusion errors with different trajectories.](Image 340x628 to 411x681)
Fuzzing Algorithm

\textbf{FusionFuzz} aims at maximizing the number of found fusion errors within a time budget. Due to the high-dimensional input space and the costly simulation execution, it is intractable to use search methods like bayesian optimization or reinforcement learning. Instead, we adopt an evolutionary-based search algorithm with a domain-specific fitness function. We denote our proposed method as GA-FUSION.

In general, our algorithm 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 \textcircled{1} in Figure 7. In \textcircled{2}, the simulation then runs \textsc{OpenPilot} in \textsc{CARLA} with the supplied scenarios. The violations found are reported and the seeds with the objective values are returned to the fuzzer accordingly. If the whole execution runs timeout, the fuzzing procedure ends (\textcircled{3}). Otherwise, seeds are ranked based on their objective values for further selection (\textcircled{4}). The fuzzer performs crossover and mutation operations among the selected seeds to generate new seeds (\textcircled{5}) for the simulation. The steps \textcircled{2}-\textcircled{5} repeat until reaching the time threshold.

The main difference between our method and previous work (Li et al. 2020; Hu et al. 2021; Ben Abdessalem et al. 2016; Abdessalem et al. 2018a) is that we focus on fusion errors rather than the general errors. Accordingly, our fitness function contains a new objective \( F_{\text{fusion}} \) for this purpose.

\textbf{Fitness Function} Given a scenario \( x \), the fitness function is a weighted sum of multiple objective functions \( F(x) = c_i \sum F_i(x) \), where \( F_i \) is an objective and \( c_i \) is the weight. Recall that fusion errors have three important properties (i.e., critical, avoidable, and fusion-induced), so we design the objective functions accordingly.

For the \textit{critical} aspect, we adopt the safety potential objective used in (Li et al. 2020). It represents the minimum value of the distance between the ego car and the leading vehicle, subtracted by the ego car’s minimum stopping distance. We denote it as \( F_d(x) \) for the scenario \( x \).

For the \textit{avoidable} aspect, we use a boolean objective function which is true only if (1) a collision happens; (2) the collided object is within the view of the ego car; and (3) the ego car is not stationary at the time the collision happens. We denote this objective function as \( F_{\text{error}} \). Note this objective also contributes to the critical aspect.

For the \textit{fusion-induced} aspect, we introduce a proxy objective. According to our definition of a fusion error, we cannot determine if an error is a fusion error without re-running the simulation with the best-sensor fusion. Alternatively, during the pre-crash period, we define \( F_{\text{fusion}} \) to be the percentage of the number of frames in which the fusion predicted lead having 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:

\[
F_{\text{fusion}}(x) = \frac{1}{|P|} \sum_{i=1}^{|P|} \left( \exists k \text{ s.t.} \| s_{ik} - y_i \|_1 < \text{th} \right)
\]

where \( P \) is a set of indices of the frames during the pre-crash period, \( K \) is a set of indices of the relevant fields of a lead, \( s_{ik} \) is a predicted lead by sensor \( k \) at time frame \( i \), \( y_i \) is the predicted lead by the fusion component, \( y_i \) is the ground-truth lead and \( \text{th} \) is an error threshold. Note each lead variable here is three-dimensional (i.e. relative longitudinal distance, relative latitudinal distance, and relative speed) and each of them is scaled down (by 4m, 1m, and 2.5m/s, respectively). \( \text{th} \) is set to 3 by default.

Finally, putting the above objectives together, we obtain the fitness function:

\[
F(x) = c_{\text{error}} F_{\text{error}}(x) + c_d F_d(x) + c_{\text{fusion}} F_{\text{fusion}}(x)
\]

Note that the objective functions are all standardized and normalized. We set the default values for \( c_{\text{error}}, c_d, c_{\text{fusion}} \) to be 1, -1, -2. Since the critical aspect essentially has two fields while the fusion aspect only has one, the choice of these default values balances the contribution of the two.

\textbf{Selection} We use binary tournament selection, as in previous work on testing ADAS (Abdessalem et al. 2018a). For each parent candidate seed, the 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 crossover.

\textbf{Crossover \& Mutation} We use simulated binary crossover (Agrawal, Deb, and Agrawal 2000) as in (Abdessalem et al. 2018a). We set the distribution index \( \eta = 5 \) and probability=0.8 to promote diversity of the offspring. We further apply polynomial mutation to each discrete and continuous variable with mutation rate set to \( \frac{1}{2} \), where \( k \) is the number of variables per instance, and the mutation magnitude \( \eta_{\text{ms}} = 5 \) to promote larger mutations.

6 Experiments

In this section, we evaluate the effectiveness of \textsc{FusionFuzz} and provide suggestions to improve the MSF methods.

6.1 Setup

\textbf{Environment.} We use \textsc{CARLA} 0.9.11 (Dosovitskiy et al. 2017) simulator and \textsc{OpenPilot} 0.8.5 (CommaAI 2021).

\textbf{Studied Fusion Methods.} We apply \textsc{FusionFuzz} on \textsc{Original} and \textsc{Mathwork} introduced in Section 5.3.

\textbf{Driving Fusion Methods.} We apply \textsc{FusionFuzz} on \textsc{Original} and \textsc{Mathwork} introduced in Section 5.3.

\textbf{Baseline and Metrics.} We leverage the random search (RANDOM) and genetic algorithm without \( F_{\text{fusion}} \) in the fitness function (GA) as two baselines. We also set the number
of specifications causing fusion errors and distinct fusion errors based on Definition 2 as two evaluation metrics.

**Hyper-parameters.** We set the pre-crash period’s m to 2.5 seconds based on empirical observations. We set s and l 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. And each simulation runs at most 20 simulation seconds.

### 6.2 The Detection Results

We compare GA-FUSION with the two baselines. Figure 8 shows the average number of fusion errors and distinct fusion errors found by the three methods over three runs for each setting. On average GA-FUSION has found 63%, 29%, 12%, 22% more fusion errors than the best baseline method under each setting, respectively. GA-FUSION has also found 45%, 22%, 14%, and 23% more distinct fusion errors (based on Definition 2) than the best baseline method, respectively. This shows the effectiveness of FusionFuzz and superiority of GA-FUSION.

![Figure 8](image)

Figure 8: # fusion errors found over # simulations. * denotes the distinct fusion errors found by that method.

![Figure 9](image)

Figure 9: Distribution density of $F_{fusion}$.

The reason the proposed GA-FUSION can find more fusion errors is that our proposed $F_{fusion}$ can differentiate fusion errors from non-fusion errors. Figure 9 shows the distribution density of $F_{fusion}$ of normal, non-fusion errors, and fusion errors when running GA-FUSION with two fusion methods. In Figure 9a fusion errors have much larger $F_{fusion}$ than non-fusion errors. The Cohen’s effect size test (Sawilowsky 2009) between them is 2.06 and the Wilcoxon rank-sum test has the p-value $1.8e^{-13}$, suggesting large difference and statistical significance, respectively. In Figure 9a fusion errors also show larger $F_{fusion}$ than non-fusion errors. The Cohen’s d effect size test (Sawilowsky 2009) between them is 0.72 and the Wilcoxon rank-sum test has the p-value $8.1e^{-3}$, meaning a medium difference and also statistical significance, respectively. Overall, the difference between fusion errors and non-fusion errors with respect to $F_{fusion}$ shows the effectiveness of $F_{fusion}$ and thus the gain of GA-FUSION.

### 6.3 Case Studies

In this subsection, we show three typical fusion errors found by FusionFuzz and analyze their root causes.

**Case1: Incorrect camera lead dominates accurate radar lead.** The first row of Figure 10 shows a failure due to the misbehavior of 2 in Figure 4a. In Figure 10a both camera and radar give accurate prediction of the leading green car at $t_0$. At $t_1$ in Figure 10b 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%) thus missing all leading vehicles due to 2 in Figure 4a. The ego car keeps driving until hitting the green car at $t_2$ in Figure 10c. If the radar data is used instead from $t_0$, however, the accident can be avoided, as shown in Figure 10d. Going back to Figure 4a the root cause is that OPENPILOT prioritizes the camera prediction and it ignores any leading vehicles if the camera prediction confidence is below 0.5. As a result, despite the accurate information predicted by the radar, OPENPILOT still causes the collision.

![Figure 10](image)

Figure 10: Two found fusion errors for ORIGINAL. rel v represents relative speed to the leading NPC vehicle.

**Case2: Inaccurate radar lead selected due to mismatch between radar and camera.** The second row of Figure 10 shows.

![Figure 10](image)

Figure 10: Two found fusion errors for ORIGINAL. rel v represents relative speed to the leading NPC vehicle.

**Case3: Inaccurate radar lead selected due to mismatch between radar and camera.** The second row of Figure 10 shows.
shows another failure caused by both 2 and 3 in Figure 4a. At t₀ of Figure 10c, camera overestimates the longitudinal distance to the leading green car. At t₁ of Figure 10f, 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 4a. 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 t₂ in Figure 10g. If the green car radar lead is used from t₀, the ego car would slow down quickly and would not hit the green car at t₂ in Figure 10h. This failure also correlates to camera dominance but it additionally involves mismatching in 3 of Figure 4a.

**Case3:** **Discarding correct lead due to a faulty selection method.** Figure 11 shows an example when MATHWORK fails due to 3 in Figure 4b. At t₀ in Figure 11a, a red car on the right lane is cutting in. While radar gives very accurate prediction data of the red car, the data not used since 3 in Figure 4b only selects among the camera leads data within the current lane. Consequently, a camera predicted lead is used, which overestimates the relative longitudinal distance. At t₁ in Figure 11b, the correct radar prediction is used but it is too late for the ego car to slow down, causing the collision at t₂ in Figure 11c. If the best predicted lead (i.e., the one from the radar data) is used starting at t₀, the collision would disappear at the time t₂ of Figure 11d.

| S1 ORIGINAL | S1 MATHWORK | S2 ORIGINAL | S2 MATHWORK |
|-------------|-------------|-------------|-------------|
| 43/52       | 14/26       | 67/78       | 13/26       |

Table 1: # avoided / # distinct fusion errors.

![Figure 12: An fusion error avoided by MATHWORK+.](image)

**6.4 Avoiding Fusion Errors**

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). MATHWORK 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 every 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 being discarded. We call this new fusion method MATHWORK+.

We evaluate MATHWORK+ by replacing the original fusion method with it during the pre-crash window on the previously found fusion errors. As shown in Section 6.4, at least 50% of found fusion errors can be avoided. Figure 12 shows a fusion error found on MATHWORK but avoided by MATHWORK+. At t₀ (Figure 12a), the ego car and a red truck drive on different lanes. At t₁, MATHWORK 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 4b). When the truck fully drives into the current lane, it is too late for the ego car to avoid the collision at t₂ (Figure 12c). If MATHWORK+ is used since Figure 12a, the truck would be considered a leading vehicle at Figure 12b and the collision would be avoided at t₂ (Figure 12d). These results demonstrate the improvement of MATHWORK+. Further, it implies that with a good fusion method, many fusion errors can be avoided without modifying the sensors or the processing units.

**7 Limitations**

There remains a gap between the real-world road testing and the simulation-based testing. However, road testing is overly expensive while simulation is flexible and efficient. Besides, we only test MATHWORK+ on limited detected fusion errors on OPENPILOT. There might be corner cases that are not covered. We leave such comprehensive study for future work. Finally, the current fusion objective only applies to high-level fusion and is only tested on two popular fusion methods. We plan to study other types of fusion methods in the 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 errors causally induced by MSF in an end-to-end system. We further propose a grey-box fuzzing framework, Fusion-Fuzz, 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 fusion methods.

**References**

Abdessalem, R. B.; Nejati, S.; Briand, L. C.; and Stifter, T. 2018a. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE.
Abdessalem, R. B.; Panichella, A.; Nejati, S.; Briand, L. C.; and Stifter, T. 2018b. Testing Autonomous Cars for Feature Interaction Failures Using Many-Objective Search. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, 143–154. New York, NY, USA: Association for Computing Machinery. ISBN 9781450359375.

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

Agrawal, R.; Deb, K.; and Agrawal, R. 2000. Simulated Binary Crossover for Continuous Search Space. Complex Systems, 9.

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

Ben Abdessalem, R.; Nejati, S.; Briand, L. C.; and Stifter, T. 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.

Cao, Y.; Wang, N.; Xiao, C.; Yang, D.; Fang, J.; Yang, R.; Chen, Q.; Liu, M. D.; and Li, B. 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.

Chen, B.; and Li, L. 2020. Adversarial Evaluation of Autonomous Vehicles in Lane-Change Scenarios. CommaAI. 2021. Openpilot.

Ding, W.; Chen, B.; Xu, M.; and Zhao, D. 2020. Learning to Collide: An Adaptive Safety-Critical Scenarios Generating Method. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Dosovitskiy, A.; Girdhar, G. 2017. CARLA: An Open Urban Driving Simulator. volume 78 of Proceedings of Machine Learning Research, 1–16. PMLR.

Ester, M.; Kriegel, H.-P.; Sander, J.; and Xu, X. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, 226–231. AAAI Press.

Hu, Z.; Guo, S.; Zhong, Z.; and Li, K. 2021. Coverage-based Scene Fuzzing for Virtual Autonomous Driving Testing. arXiv.

Koren, M.; Alsaif, S.; Lee, R.; and Kochenderfer, M. J. 2018. Adaptive Stress Testing for Autonomous Vehicles. In 2018 IEEE Intelligent Vehicles Symposium (IV), 1–7.

Kuutti, S.; Fallah, S.; and Bowden, R. 2020. Training Adversarial Agents to Exploit Weaknesses in Deep Control Policies. Li, G.; Li, Y.; Jha, S.; Tsai, T.; Sullivan, M.; Hari, S. K. S.; Kalbarczyk, Z.; and Iyer, R. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 25–36.

Ma, Y.; Sharp, J. A.; Wang, R.; Fernandes, E.; and Zhu, X. 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, 8865–8873. AAAI Press.

Manes, V.; Han, H.; Han, C.; Egele, M.; Schwartz, E. J.; and Woo, M. 2019. The Art, Science, and Engineering of Fuzzing: A Survey. IEEE Transactions on Software Engineering, (01): 1–1.

Mathwork. 2021. Forward Collision Warning Using Sensor Fusion.

Menzies, P.; and Beebee, H. 2020. Counterfactual Theories of Causation. In Zalta, E. N., ed., The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Winter 2020 edition.

Norden, J.; O’Kelly, M.; and Sinha, A. 2019. Efficient Black-box Assessment of Autonomous Vehicle Safety. In Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019).

O’Kelly, M.; Sinha, A.; Namkoong, H.; Tedrake, R.; and Duchi, J. C. 2018. Scalable End-to-End Autonomous Vehicle Testing via Rare-event Simulation. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesani-Bianchi, N.; and Garnett, R., eds., Advances in Neural Information Processing Systems, volume 31, 9827–9838. Curran Associates, Inc.

R. M.; A. M.; L. S.; and D. Z. 2021. Paracosm: A Test Framework for Autonomous Driving Simulations. Fundamental Approaches to Software Engineering.

Sawilowsky, S. 2009. New Effect Size Rules of Thumb. Journal of Modern Applied Statistical Methods, 8: 26.

Shen, J.; Won, J.; Chen, Z.; and Chen, Q. A. 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.

Stuff, A. 2019. Delphi Electronically Scanning RADAR. TheAutowareFoundation. 2016. Autoware: Open-source software for urban autonomous driving.

Tu, J.; Li, H.; Yan, X.; Ren, M.; Chen, Y.; Liang, M.; Bitar, E.; Yumer, E.; and Urtasun, R. 2021. Exploring Adversarial Robustness of Multi-Sensor Perception Systems in Self Driving. Wenhao, D.; Baiminng, C.; Bo, L.; Kim, J. E.; and Ding, Z. 2020. Multimodal Safety-Critical Scenarios Generation for Decision-Making Algorithms Evaluation. In arXiv.

Wheeler, T. A.; and Kochenderfer, M. J. 2019. Critical Factor Graph Situation Clusters for Accelerated Automotive Safety Validation. In 2019 IEEE Intelligent Vehicles Symposium (IV), 2133–2139.
Yeong, D. J.; Velasco-Hernandez, G.; Barry, J.; and Walsh, J. 2021. Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. Sensors, 21(6).

Zhong, Z.; Kaiser, G.; and Ray, B. 2021. Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles. arXiv preprint arXiv:2109.06126.

Zhou, T.; Yang, M.; Jiang, K.; Wong, H.; and Yang, D. 2020. MMW Radar-Based Technologies in Autonomous Driving: A Review. Sensors, 20(24).
A Adaption of Mathwork Radar-Camera Fusion
The original Mathwork implementation is in Matlab, we reimplement it in Python. Besides, the Mathwork implementation uses an Extended Kalman Filter since the radar measurements are non-linear. Since OPENPILOT takes a transformed linear radar measurements (i.e. the relative position is in the Cartesion coordinate rather than angular coordinate), we use a Linear Kalman Filter instead. Besides, the speed of $y$ axis (in terms of the ego car’s coordinate) is not supported by the OPENPILOT interface so we leave this field as a constant in the Kalman Filter modeling. These changes can potentially introduce fusion errors that may not appear in the original implementation.

B Modifications on OPENPILOT and CARLA
B.1 Settings for CARLA and Changes made on OPENPILOT for Better Reproducibility
By default, CARLA client (controlling the auto-driving car), traffic manager (controlling NPC vehicles), and server (control the world environment) use asynchronous communication. The CARLA world also uses a variable time step so the virtual time between two updates of the world is equal to the real-world computing time. Besides, OPENPILOT is a real-time system and its communication with CARLA is asynchronous. These default designs make reproducing an accident of OPENPILOT in CARLA very difficult. To conquer these difficulties, we use the following parameter settings and modifications.

First, we adopt synchronous mode for CARLA and CARLA TRAFFIC MANAGER by setting random seed to be 0 for CARLA TRAFFIC MANAGER, and use fixed virtual time step of 0.01 seconds. These allow the scenarios created to be deterministic. Second, inspired by some design choices of Testpilot(Norden, O’Kelly, and Sinha 2019) which was developed based on OPENPILOT 0.5 achieving complete synchronization between CARLA and OPENPILOT, we made changes that are enough for fusion errors reproducibility rather than complete synchronization (very difficult since many changes have happend since OPENPILOT 0.5). In particular, we modify OPENPILOT to use the simulator’s virtual time passed from the bridge between the simulator and the controller rather than using real time. Note that if OPENPILOT uses real time, CARLA will be not able to catch up with the speed of OPENPILOT and thus lead to significant communication delay. What’s more, before each scenario starts, we allow OPENPILOT to take in sensor input from CARLA for 4 virtual seconds to allow all of its modules up and running. This step is necessary since on different machines, it takes different virtual time for OPENPILOT to successfully start all the modules due to potential delay of message passing across modules. With these changes, nearly all fusion errors can be reproduced across runs and machines. Because of all these changes made, although we try the best to make reasonable changes, it is still possible that some of the crashes reported here may not happen on a specific vehicle controlled by OPENPILOT in the real world.

B.2 Radar Implementation for OPENPILOT in CARLA
The official OPENPILOT bridge implementation does not implement radar in the simulation environment. To evaluate the fusion component of OPENPILOT, we create a radar sensor in CARLA following Delphi Electronically Scanning RADAR (Stuff 2019) (174m range and 30 horizontal degrees) and send processed radar data at 20Hz. Since the radar interface in CAN of OPENPILOT takes in pre-processed 16 tracks including relative x, relative y, and relative speed, the raw radar information in CARLA is pre-processed using DBSCAN (Ester et al. 1996) (with eps= 0.5 and min samples= 5), a widely used for raw radar data pre-processing (Zhou et al. 2020).

C Search Space Details
Both driving environments have 6 NPC vehicles. The maximum allowed speed of the auto-driving car is set to 45 miles/hr on the highway road (S2) and 35 miles/hr on the local road (S1). The search space for each vehicle consists of vehicles type (including truck, bicyclist, motorcycle, and multiple car types), its speed (from 0 to 50% beyond the maximum allowed speed of the current road) and lane change decision (turn left/right, or stay in lane) at each time interval. Lighting condition is controlled by sun azimuth angle and sun altitude angle. Weather consists of the following fields: cloudiness, precipitation, precipitation deposits, wind intensity, fog density, fog distance, wetness, and fog falloff, the detailed explanation for each field can be found on CARLA Python API (Dosovitskiy et al. 2017). The search space is $76 (= 6 + 6 \times 2 \times 5 + 2 + 8)$ in total.

D Failure of using Radar Dominance for MSF
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 6.3 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.

E More on Experiment Environment and Hyper-parameters
All the experiments run on a Ubuntu20.04 desktop with Intel i9-7940X, Nvidia 2080Ti, and 32GB memory. The random seeds are set to 0, 10, 20 respectively for the three runs of each fuzzing method.