

University of Wollongong

Research Online

Faculty of Engineering and Information
Sciences - Papers: Part B

Faculty of Engineering and Information
Sciences

2019

Metamorphic Testing of Driverless Cars

Zhi Q. Zhou

University of Wollongong, zhiquan@uow.edu.au

Liqun Sun

University of Wollongong, ls168@uowmail.edu.au

Follow this and additional works at: <https://ro.uow.edu.au/eispapers1>



Part of the [Engineering Commons](#), and the [Science and Technology Studies Commons](#)

Recommended Citation

Zhou, Zhi Q. and Sun, Liqun, "Metamorphic Testing of Driverless Cars" (2019). *Faculty of Engineering and Information Sciences - Papers: Part B*. 2384.

<https://ro.uow.edu.au/eispapers1/2384>

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Metamorphic Testing of Driverless Cars

Abstract

On March 18, 2018, Elaine Herzberg became the first pedestrian in the world to be killed by an autonomous vehicle after being hit by a self-driving Uber SUV in Tempe, AZ, at about 10 p.m. Video released by the local police department showed the self-driving Volvo XC90 did not appear to see Herzberg, as it did not slow down or alter course, even though she was visible in front of the vehicle prior to impact. Subsequently, automotive engineering experts raised questions about Uber's LiDAR technology.¹² LiDAR, or "light detection and ranging," uses pulsed laser light to enable a self-driving car to see its surroundings hundreds of feet away.

Disciplines

Engineering | Science and Technology Studies

Publication Details

Zhou, Z.Q. & Sun, L. (2019). Metamorphic Testing of Driverless Cars. *Communications of the ACM*, 62 (3), 61-67.

Metamorphic Testing of Driverless Cars

ZHI QUAN ZHOU and LIQUN SUN, University of Wollongong, Australia

<http://dx.doi.org/10.1145/3241979> (to become available after March 2019)

1 THE FATAL ACCIDENT

On March 18, 2018, Elaine Herzberg became the first pedestrian to be killed by an autonomous vehicle after being hit by a self-driving Uber SUV in Tempe, Arizona, at about 10 pm. Footage released by police showed that the self-driving Volvo XC90 did not appear to see Herzberg—the car did not slow down or alter its course even though the pedestrian was visible in front of the vehicle prior to the collision. Subsequently, experts raised questions about Uber’s LiDAR technology [12]. LiDAR stands for “Light Detection and Ranging,” which uses pulsed laser light to enable the car to see surroundings hundreds of feet away.

The supplier of the Uber vehicle’s LiDAR technology, Velodyne, said that “our LiDAR is capable of clearly imaging Elaine and her bicycle in this situation. However, our LiDAR doesn’t make the decision to put on the brakes or get out of her way,” “we know absolutely nothing about the engineering of their [Uber’s] part . . . It is a proprietary secret and all of our customers keep this part to themselves” [15], and that “our LiDAR can see perfectly well in the dark, as well as it sees in daylight, producing millions of points of information. However, it is up to the rest of the system to **interpret and use** the data to make decisions. We do not know how the Uber system of decision-making works” [11].

2 A QUESTION CONCERNING EVERY HUMAN LIFE

Regardless of the investigation outcomes, the Uber fatal accident raised a serious question concerning the perception capability of self-driving cars:

Are there situations where a driverless car’s on-board computer system could incorrectly “interpret and use” the data sent from a sensor such as a LiDAR sensor, making the car unable to detect a pedestrian or an obstacle on the roadway?

This question is not specific to Uber cars, but is general enough to cover all types of autonomous vehicles, and the answer to this question concerns every human life. Our answer to this question is, unfortunately, **affirmative**. Even though we could not access the Uber system, we have managed to test Baidu Apollo, the well-known real-world self-driving software controlling many autonomous vehicles on the road (<http://apollo.auto>). Using a novel *metamorphic testing* method, we have detected critical software errors that could cause the Apollo’s *perception* module to misinterpret

Authors’ address: Zhi Quan Zhou; Liqun Sun, Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

Manuscript submitted to ACM

Manuscript submitted to ACM

the *point cloud* data sent from the LiDAR sensor, making some pedestrians and obstacles undetectable. The Apollo system uses Velodyne’s HDL64E LiDAR sensor [1], exactly the same type of LiDAR involved in the Uber accident [16].

We reported this issue to the Baidu Apollo self-driving car team on March 10, 2018, MST (UTC -7), *eight days before the Uber accident*. Our bug report was logged online as issue #3341 (<https://github.com/ApolloAuto/apollo/issues/3341>). We did not receive a response from Baidu until 10:25 pm, March 19, 2018, MST—24 hours after the Uber accident. In the reply, the Apollo perception team confirmed the error.

Before presenting further details of our findings, we will first discuss the challenges for testing complex computer systems, with a focus on software testing for autonomous cars.

3 THE TESTING CHALLENGE

Testing is a major approach to software quality assurance. Deploying inadequately tested software can have serious consequences [22]. Software testing, however, is fundamentally challenged by the *oracle problem*. An *oracle* is a mechanism that testers use to determine whether the outcomes of test case executions are correct [2, 24]. Most software testing techniques assume that an oracle exists. This assumption, however, does not always hold when testing complex applications. This is known as the oracle problem, a situation where an oracle is unavailable or is too expensive to be applied. For example, when testing a compiler, determining the equivalence between the source code and the compiler-generated object program is difficult. When testing a web search engine, it is very hard for the tester to assess the completeness of the search results.

To achieve a high standard of testing, the tester needs to generate, execute, and verify a large number of tests. These tasks can hardly be accomplished without test automation. For the testing of autonomous vehicles, however, it is hard to construct a fully automated test oracle. Although, in some situations, the human tester could serve as an oracle, manual monitoring is expensive and error-prone. In the Uber accident, for example, the safety driver was not doing any safety monitoring, and this is not surprising because “humans monitoring an automated system are likely to become bored and disengaged,” which makes this kind of testing “particularly dangerous” [12].

The oracle problem is also reflected by the difficulty in creating detailed system specifications against which the autonomous car’s behavior can be checked, as it essentially involves recreating the logic of a human driver [21]. Besides, even for highly qualified human testers with full system specifications, it can still be difficult or even impossible for them to determine the correctness of every behavior of an autonomous vehicle. For example, in a complex road network, it is hard for the tester to decide whether the driving route selected by the autonomous car is optimal [3]. Similarly, it is not easy to verify whether the software system has correctly interpreted the huge amounts of *point cloud* data sent from a LiDAR sensor, normally at a rate of more than a million data points per second.

Even more challenging is *negative testing*: While *positive testing* focuses on ensuring that a program does what it is supposed to do for normal inputs, negative testing serves to ensure that the program does not do what it is not supposed to do when the inputs are unexpected, normally involving some random factors or events. Resource constraints and deadline pressures often result in development organizations skipping negative testing, potentially allowing safety and security issues to persist into the released software [5, 22].

In the context of negative software testing for autonomous vehicles (if ever attempted by the development organization), how can the tester identify the conditions under which the vehicle could potentially do something wrong (for example, striking a pedestrian)? To a certain degree, tools called *fuzzers* could help to perform this kind of negative software testing. During *fuzzing*, or *fuzz testing*, the fuzzer generates a random or semi-random input, and feeds it into the system under test, hoping to crash the system or cause it to misbehave [22]. However, the oracle problem makes

the verification of the fuzz test results (outputs for millions of random inputs) extremely difficult, if not impossible [5]. Therefore, in fuzzing, the tester looks only for software crashes (such as aborts and hangs). This not only means that huge amounts of test cases might need to be run before a crash, but also means that logic errors, which do not crash the system but instead produce incorrect output, cannot be detected [5]. For example, fuzzing cannot detect the error when a calculator returns “ $1 + 1 = 3$.” Neither can simple fuzzing detect misinterpretation of LiDAR data.

4 METAMORPHIC TESTING (MT)

Metamorphic testing (MT) [6] is a property-based software testing technique that can effectively address two fundamental problems in software testing: the oracle problem and automated test case generation problem. The main difference between MT and other testing techniques is that the former does not focus on the verification of each individual output of the software under test (and therefore can be performed in the absence of an oracle). MT checks the *relations* among the inputs and outputs of *multiple* executions of the software. Such relations are called *metamorphic relations* (MRs), which are necessary properties of the intended program’s functionality. If, for certain test cases, an MR is violated, then the software must be faulty. Consider, for example, the testing of a search engine. Suppose that the tester entered a search criterion C_1 and that the search engine returned 50,000 results. It may not be easy to verify the accuracy and completeness of these 50,000 results. Nevertheless, an MR can be identified as follows: The search results for C_1 must include those for $C_1 \text{ AND } C_2$, where C_2 can be any additional condition (such as a string or a filter). If the actual search results violate this relation, the search engine must be at fault. Here, the search criterion “ $C_1 \text{ AND } C_2$ ” is a new test case that can be automatically constructed based on the source test case “ C_1 ” (whereas C_2 could be generated automatically and randomly) and the satisfaction of the MR can also be automatically verified using a testing program.

A growing body of research from both industry and the academia has examined the concept of MT and proved it to be highly effective [4, 7–9, 13, 18–20]. The increasing interest in MT is not only because it can alleviate the oracle problem and automate test generation, but also because the perspective of MT has seldom been used in previous testing strategies and, as a result, MT has detected a large number of previously unknown faults in many mature systems such as the GCC and LLVM compilers [10, 17], the web search engines Google and Bing [25], and code obfuscators [5].

5 MT FOR TESTING AUTONOMOUS MACHINERY

Several research groups have started to apply MT to alleviate the difficulties in testing autonomous systems, yielding encouraging results.

Researchers from the Fraunhofer Center for Experimental Software Engineering, Maryland, USA, developed a simulated environment in which the control software of autonomous drones was tested using MT [14]. Their MRs made use of geometric transformations, such as rotation and translation, in combination with different formations of obstacles in the flying scenarios of the drone. They looked for behavioral differences of the drone when it was flying under these different (but supposedly equivalent) scenarios. Their MRs required that the drone should have consistent behavior; however, they found that in some situations the drone behaved inconsistently, revealing multiple software defects. For example, one of the bugs was in the sense-and-avoid algorithm, which made the algorithm sensitive to certain numerical values and hence misbehave under certain conditions, causing the drone to crash. They detected another bug after running hundreds of tests using different rotations of the environment: The drone had landing problems in some situations. This was because they rotated all the objects in the environment but not the sun, and this unexpectedly caused a shadow to fall on the landing pad in some orientations, revealing issues in the drone’s vision system. The researchers solved this issue by using a more robust vision sensor that was less sensitive to lighting changes.

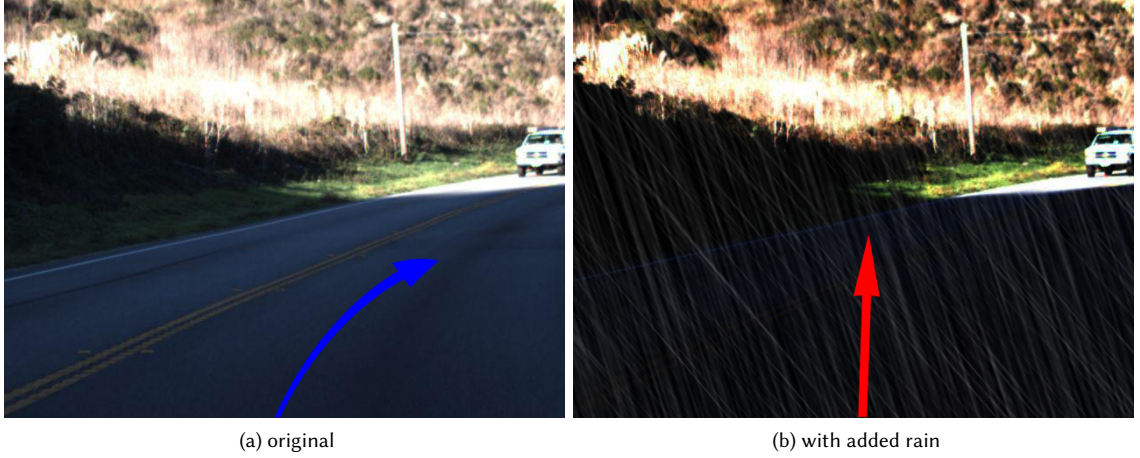


Fig. 1. A pair of input pictures used in a metamorphic test, revealing inconsistent and erroneous behavior of a DNN (<https://deeplearningtest.github.io/deepTest> [21]).

Researchers from the University of Virginia and Columbia University tested three different Deep Neural Network (DNN) models for autonomous driving [21]. The inputs to the models are pictures from a camera and the outputs are steering angles. To verify the correctness of the outputs, the researchers used a set of MRs based on image transformations. The MRs stated that the car should behave similarly for variations of the same input (for example, the same scene under different lighting conditions). Using these MRs, they generated realistic synthetic images based on seed images. These synthetic images mimic real-world phenomena such as camera lens distortions and different weather conditions. Using MT, together with a notion of neuron coverage (the number of neurons activated), the researchers found a large number of corner case inputs leading to erroneous behavior in three DNN models. Figure 1 shows such an example, where the original trajectory (the blue arrow) and the second trajectory (the red arrow) are inconsistent, revealing dangerous erroneous behavior of the DNN model under test.

Recently, we have applied MT to test Google Maps services that can be used to navigate autonomous cars [3]. We identified a set of MRs for the navigation system. For example, one of the MRs is stated as “a restrictive condition such as avoiding tolls should not result in a more optimal route.” Using these MRs, we have detected a large number of real-life bugs in Google Maps, one of which is shown in Figure 2: The origin and destination points were almost at the same location; however, Google Maps returned a route of 4.9 miles, which was obviously unacceptable.

6 THE DETECTION OF REAL-LIFE LIDAR DATA INTERPRETATION ERRORS

The scope of the study conducted by the researchers from the University of Virginia and Columbia University [21] was limited to DNN models only. A DNN model is but part of the perception module of a self-driving car’s software system. Furthermore, while a DNN can take input from different sensors such as camera and LiDAR, they only studied the ordinary 2D picture input from a camera, and the output considered was the steering angle calculated by the DNN model based on the input picture. The software that they tested was *not* real-life systems controlling autonomous cars, but instead was deep learning models that “won top positions in the Udacity self-driving challenge.”

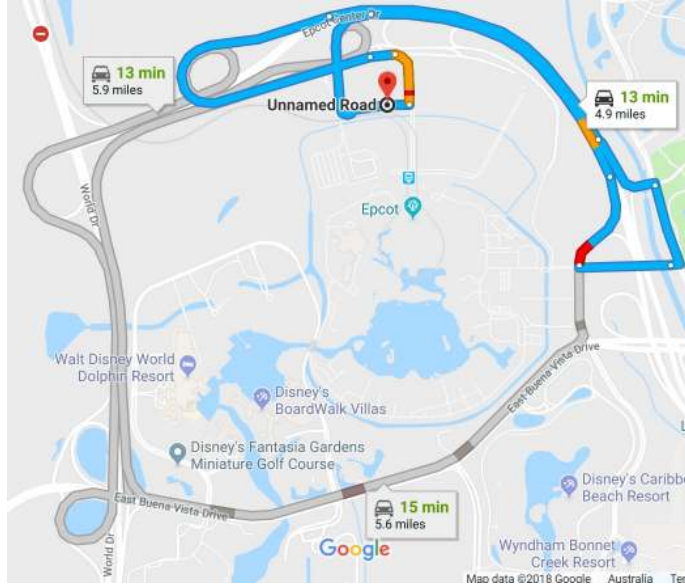


Fig. 2. MT detected a real-life bug in Google Maps [3, 20]: The origin and destination were almost at the same point but Google Maps generated an “optimal” route of 4.9 miles.

In contrast to their work, this article reports our testing of the real-life system Apollo, which is the on-board software of Baidu’s self-driving vehicles. Baidu also claims that users can directly use this software to build their own autonomous cars (http://apollo.auto/cooperation/detail_en_01.html).

6.1 The Software Under Test

More specifically, we tested Apollo’s *perception* module (<http://apollo.auto/platform/perception.html>), which has two key components: 3D Obstacle Perception, and Traffic Light Perception. We tested the 3D Obstacle Perception component, which consisted of three subsystems: LiDAR Obstacle Perception, RADAR Obstacle Perception, and Obstacle Results Fusion. Although our testing method is applicable to all three subsystems, we only tested the first subsystem, LiDAR Obstacle Perception (this subsystem is referred to as LOP hereafter). LOP takes the 3D point cloud data as input, generated by Velodyne’s HDL64E LiDAR sensor.

LOP resolves the raw point cloud data using the following pipeline (the following are excerpts of the Apollo website https://github.com/ApolloAuto/apollo/blob/master/docs/specs/3d_obstacle_perception.md):

- (1) HDMap Region of Interest (ROI) Filter (**tested in our experiments**): The Region of Interest (ROI) specifies the drivable area that includes road surfaces and junctions that are retrieved from the HD (high-resolution) map. The HDMap ROI filter processes LiDAR points that are outside the ROI, removing background objects, e.g., buildings and trees around the road. What remains is the point cloud in the ROI for subsequent processing.
- (2) Convolutional Neural Networks (CNN) Segmentation (**tested in our experiments**): After identifying the surrounding environment using the HDMap ROI filter, Apollo obtains the filtered point cloud that includes only the points inside the ROI (i.e., the drivable road and junction areas). Most of the background obstacles, such as buildings and trees around the road region, have been removed, and the point cloud inside the ROI is fed into the

segmentation module. This process detects and segments out foreground obstacles, e.g., cars, trucks, bicycles, and pedestrians. Apollo uses a deep CNN for accurate obstacle detection and segmentation. The output of this process is a set of objects corresponding to obstacles in the ROI.

- (3) MinBox Builder (**tested in our experiments**): This object builder component establishes a bounding box for the detected obstacles. The main purpose of the bounding box is to estimate the heading of the obstacle (e.g., vehicle) even if the point cloud is sparse. Equally, the bounding box is used to visualize the obstacles.
- (4) HM Object Tracker (**not tested in our experiments**): This tracker is designed to track obstacles detected by the segmentation step.
- (5) Sequential Type Fusion (**not tested in our experiments**): To smooth the obstacle type and reduce the type switch over the entire trajectory, Apollo utilizes a sequential type fusion algorithm.

Our software testing experiments involved (1), (2) and (3), but not (4) and (5), because the first three are the most critical and fundamental features.

6.2 Our Testing Method: MT in Combination with Fuzzing

Based on the Baidu specification of the *HDMaP Region of Interest (ROI) Filter* described above, we identified the following metamorphic relation, where the software under test is the LiDAR Obstacle Perception subsystem (LOP), A and A' denote two inputs to LOP, and O and O' denote LOP's outputs for A and A' , respectively:

MR_1 : Let A and A' be two frames of 3D point cloud data that are identical except that A' includes a small number of additional LiDAR data points randomly scattered in regions outside the ROI. Let O and O' be the sets of obstacles identified by LOP for A and A' , respectively (LOP only identifies obstacles within the ROI). Then, the following relation must hold: $O \subseteq O'$.

In MR_1 , the additional LiDAR data points in A' could represent small particles in the air, or could just be some noise from the sensor, whose existence is very possible [23]. MR_1 states that the existence of some particles, or some noise points, or their combination, in the air far away outside the ROI should not cause an obstacle on the roadway to become undetectable. As an extreme example, a small insect 100 meters away outside the ROI should not affect the detection of a pedestrian in front of the vehicle. This requirement is intuitively valid and agrees with the Baidu specification of the HDMaP ROI Filter. In fact, we could also require that $O = O'$. According to the user manual, the HDL64E, which can be mounted on top of the vehicle, delivers a 360° horizontal field of view and a 26.8° vertical field of view, capturing a point cloud with a range of up to 120 meters.

Next, we describe the design of three series of experiments to test the LOP using MR_1 . The Apollo Data Open Platform (<http://data.apollo.auto>) provides a set of “Vehicle System Demo Data”—the sensor data collected at real scenes. We downloaded the main file of this dataset, named demo-sensor-demo-apollo-1.5.bag (8.93GB). This file included point cloud data collected by Baidu engineers using the Velodyne LiDAR sensor in the morning of September 6, 2017. In each series of experiments, we first randomly extracted 1000 frames of the point cloud data—we call each frame a *source test case*. For each source test case t , we ran the LOP software to identify its ROI and to generate O , the set of detected obstacles for t . We then constructed a *follow-up test case* t' by randomly scattering n additional points into the 3D space outside the ROI of t —the value of the z coordinate of each point was determined by choosing a random value between the minimum and maximum z -coordinate values of all points in t . Using a similar approach, we also generated a d value (the reflected intensity of the laser) for each added point. The d value was needed for each point, in addition to the 3D coordinates. We then ran the LOP software for t' , producing O' , the set of detected obstacles. Finally, we compared O

Table 1. Summary of test results (for each value of n , 1000 pairs of results were compared).

number of added points (n)	$ O > O' $	$ O = O' $	$ O < O' $
10	27	951	22
100	121	781	98
1000	335	533	132

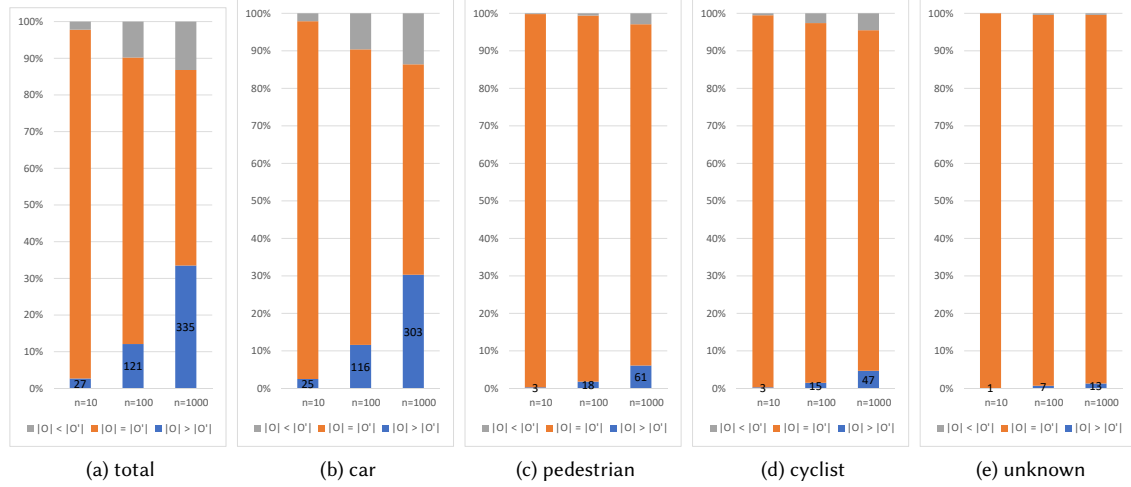


Fig. 3. Results of experiments by category: 100% stacked column charts. Each vertical column represents the comparisons of 1000 pairs of results, where the blue subsection implies MR_1 violations. Each blue subsection is labeled with the actual number of $|O| > |O'|$ cases (out of the 1000 pairs).

and O' . We conducted three series of experiments for $n = 10, 100, 1000$. Therefore, the LOP software was run for a total of $(1000 + 1000) \times 3 = 6000$ times, processing 3000 source test cases and 3000 follow-up test cases.

6.3 Test Results

In our experiments, for ease of implementation of MR_1 , we did not check the subset relation $O \subseteq O'$, but instead we compared the numbers of objects contained in O and O' , denoted by $|O|$ and $|O'|$, respectively. Note that $O \subseteq O' \rightarrow |O| \leq |O'|$ (hence, the condition we actually checked was less strict than MR_1). In other words, if $|O| > |O'|$ then there must be something wrong, as one or more objects in O must be missing in O' .

The results of experiments are quite surprising. Table 1 summarizes the overall results. The violation rates (that is, cases for $|O| > |O'|$ out of 1000 pairs of outputs) are 2.7% ($= 27 \div 1000$), 12.1% ($= 121 \div 1000$), and 33.5% ($= 335 \div 1000$), for $n = 10, 100$, and 1000, respectively. This means that, as few as 10 sheer random points dispersedly scattered in the vast 3D space outside the ROI can cause the autonomous car to fail to detect an obstacle on the roadway, with a 2.7% probability. When the number of random points increases to 1000, the probability becomes as high as 33.5%. According to the HDL64E user manual, the LiDAR sensor generates more than a million data points per second, and each frame of point cloud data used in our experiments normally contained more than 100,000 data points. As thus, the random points we added to the point cloud frames are trivial.

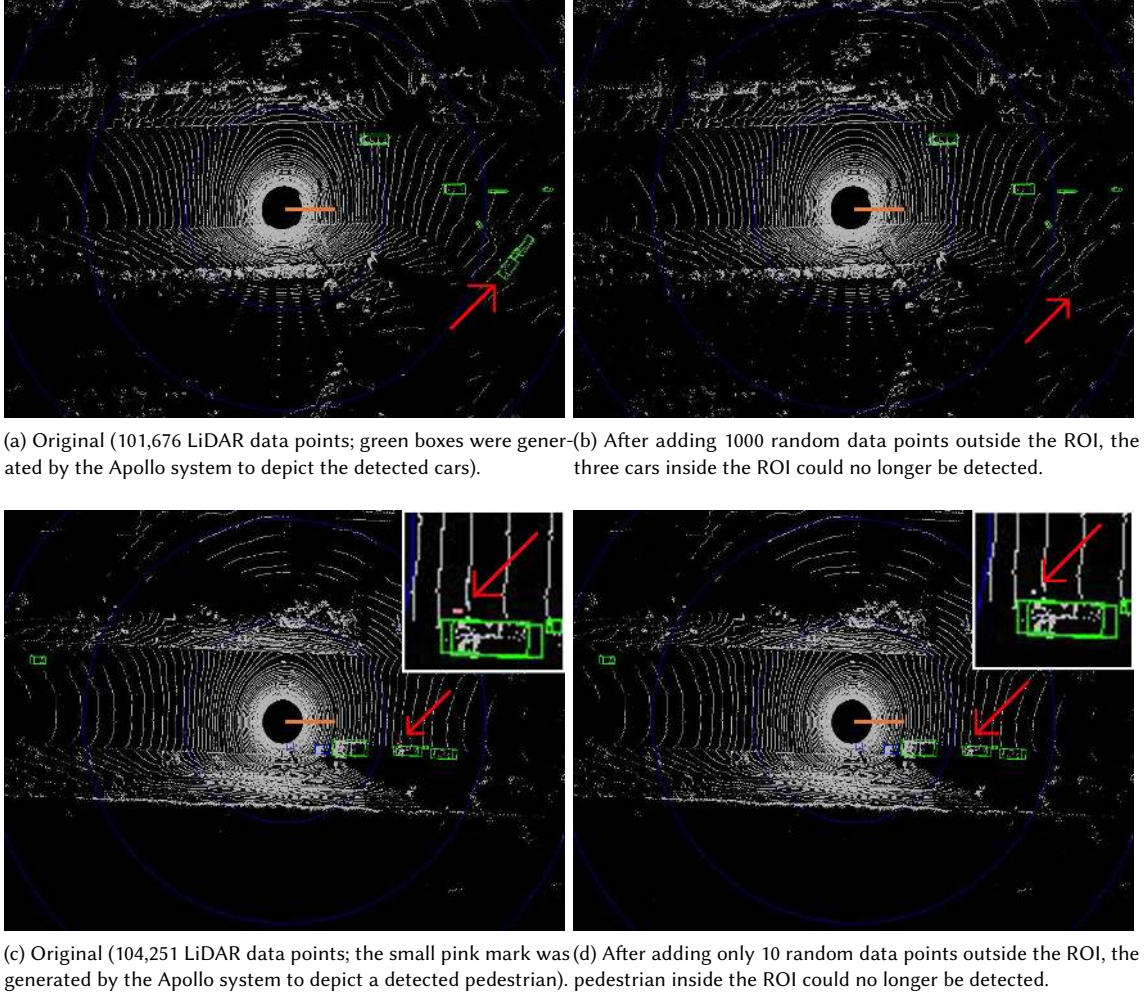



Fig. 4. MT detected real-life fatal errors in LiDAR point cloud data interpretation within the Apollo *perception* module: three missing cars and one missing pedestrian.

The LOP software categorizes the detected obstacles into four types: “detected car, pedestrian, cyclist and unknown are depicted by bounding boxes in green, pink, blue and purple respectively” (<http://apollo.auto/platform/perception.html>). Figures 3b to 3e summarize the test results of these four categories, and Figure 3a shows the overall results corresponding to Table 1.

Each vertical column of Figure 3 has a subsection in blue, corresponding to MR_1 violations. These blue subsections are labeled with the actual numbers of $|O| > |O'|$ cases. It is observed that all these numbers are greater than 0, indicating critical errors in the perception of all four types of obstacles: car, pedestrian, cyclist and unknown. Relatively speaking, the error rate of the “car” category is the highest, followed by “pedestrian,” “cyclist,” and “unknown.”

Figures 4a and 4b show a real-world example revealed by our test, where three cars inside the ROI could not be detected after 1000 random points were added outside the ROI. Figures 4c and 4d show another example, where a pedestrian inside the ROI (the Apollo system depicted this pedestrian using the small pink mark  as shown in Figure 4c) could not be detected after only 10 random points were added outside the ROI (as shown in Figure 4d, the small pink mark is missing). As mentioned in Section 2, we reported the bug to the Baidu Apollo self-driving car team on March 10, 2018 (<https://github.com/ApolloAuto/apollo/issues/3341>). On March 19, 2018, the Apollo team confirmed the error by acknowledging that “it might happen” and suggesting that “for cases like that, models can be fine tuned using data augmentation”—data augmentation is a technique that alleviates the problem of lack of training data in machine learning by inflating the training set through transformations of the existing data. Obviously, our failure-causing metamorphic test cases (those with the random points) could serve the purpose.

7 CONCLUSION

The Uber fatal crash revealed inadequacy of conventional testing approaches for mission-critical autonomous systems. We have shown that MT alleviates this problem and enables automatic detection of fatal errors in self-driving machinery that operates on either conventional algorithms or deep learning models. We have introduced an innovative testing strategy that combines MT with fuzzing, and reported how we used this strategy to detect previously unknown fatal errors in the real-life LiDAR Obstacle Perception system of the Baidu Apollo self-driving software.

The scope of this study is limited to LiDAR obstacle perception. Apart from LiDAR, an autonomous machinery may also be equipped with radar. According to the Apollo website (<http://data.apollo.auto>), “radar could precisely estimate the velocity of moving obstacles, while LiDAR point cloud could give a better description of object shape and position.” Furthermore, there can also be cameras, which are particularly useful for the detection of visual features such as the color of traffic lights. Our testing technique can be equally applied to radar, camera, and other types of sensor data, as well as obstacle fusion algorithms involving multiple sensors. In future research, we plan to collaborate with industry to develop MT-based testing techniques, in combination with existing verification and validation methods, to make driverless cars safer.

ACKNOWLEDGMENTS

This work was supported in part by a linkage grant of the Australian Research Council (project ID: LP160101691). We would like to thank Suzhou Insight Cloud Information Technology Co., Ltd. for supporting this research.

REFERENCES

- [1] Baidu, Inc. March 2018. Apollo Reference Hardware. <http://apollo.auto/platform/hardware.html>. (March 2018).
- [2] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525.
- [3] Joshua Brown, Zhi Quan Zhou, and Yang-Wai Chow. 2018. Metamorphic testing of navigation software: A pilot study with Google Maps. In *Proceedings of the 51st Annual Hawaii International Conference on System Sciences (HICSS-51)*. 5687–5696. Available: <http://hdl.handle.net/10125/50602>.
- [4] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys* 51, 1 (2018), 4:1–4:27.
- [5] Tsong Yueh Chen, Fei-Ching Kuo, Wenjuan Ma, Willy Susilo, Dave Towey, Jeffrey Voas, and Zhi Quan Zhou. 2016. Metamorphic testing for cybersecurity. *Computer* 49, 6 (2016), 48–55.
- [6] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. 2003. Fault-based testing without the need of oracles. *Information and Software Technology* 45, 1 (2003), 1–9.
- [7] Alastair F. Donaldson, Hugues Evrard, Andrei Lascu, and Paul Thomson. 2017. Automated Testing of Graphics Shader Compilers. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 93:1–93:29.

- [8] Darryl C. Jarman, Zhi Quan Zhou, and Tsong Yueh Chen. 2017. Metamorphic testing for Adobe data analytics software. In *Proceedings of the IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET '17), in conjunction with the 39th International Conference on Software Engineering (ICSE '17)*. 21–27. <https://doi.org/10.1109/MET.2017.1>
- [9] Upulee Kanewala, Laura L. Pullum, Sergio Segura, Dave Towey, and Zhi Quan Zhou. 2016. Message from the workshop chairs. In *Proceedings of the IEEE/ACM 1st International Workshop on Metamorphic Testing (MET '16), in conjunction with the 38th International Conference on Software Engineering (ICSE '16)*. ACM Press.
- [10] Vu Le, Mehrdad Afshari, and Zhendong Su. 2014. Compiler validation via equivalence modulo inputs. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)*. 216–226.
- [11] Dave Lee. March 23, 2018. Sensor firm Velodyne 'baffled' by Uber self-driving death. <http://www.bbc.com/news/technology-43523286>. (March 23, 2018).
- [12] Sam Levin. March 23, 2018. Uber crash shows 'catastrophic failure' of self-driving technology, experts say. <https://www.theguardian.com/technology/2018/mar/22/self-driving-car-uber-death-woman-failure-fatal-crash-arizona>. (March 23, 2018).
- [13] Mikael Lindvall, Dharmalingam Ganesan, Ragnar Årdal, and Robert E. Wiegand. 2015. Metamorphic model-based testing applied on NASA DAT – an experience report. In *Proceedings of the IEEE/ACM 37th International Conference on Software Engineering (ICSE '15)*. 129–138.
- [14] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. 2017. Metamorphic model-based testing of autonomous systems. In *Proceedings of the IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET '17), in conjunction with the 39th International Conference on Software Engineering (ICSE '17)*. 35–41.
- [15] Alan Ohnsman. March 23, 2018. LiDAR maker Velodyne 'baffled' by self-driving Uber's failure to avoid pedestrian. <https://www.forbes.com/sites/alanohnsman/2018/03/23/lidar-maker-velodyne-baffled-by-self-driving-ubers-failure-to-avoid-pedestrian>. (March 23, 2018).
- [16] Matt Posky. March 23, 2018. LIDAR supplier defends hardware, blames Uber for fatal crash. <http://www.thetruthaboutcars.com/2018/03/lidar-supplier-blames-uber/>. (March 23, 2018).
- [17] John Regehr. June 20, 2014. Finding Compiler Bugs by Removing Dead Code. <http://blog.regehr.org/archives/1161>. (June 20, 2014).
- [18] Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on Software Engineering* 42, 9 (2016), 805–824.
- [19] Sergio Segura and Zhi Quan Zhou. 2017. Metamorphic Testing: Introduction and Applications. <https://event.on24.com/wcc/r/1451736/8B5B5925E82FC9807CF83C84834A6F3D>. (2017). ACM SIGSOFT Webinar.
- [20] Sergio Segura and Zhi Quan Zhou. 2018. Metamorphic testing 20 years later: A hands-on introduction. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE '18)*. ACM.
- [21] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE '18)*.
- [22] Apostol Vassilev and Christopher Celi. 2014. Avoiding cyberspace catastrophes through smarter testing. *Computer* 47, 10 (October 2014), 102–106.
- [23] White Paper. 2007. Velodyne's HDL-64E: A High Definition LIDAR Sensor for 3-D Applications. www.velodynelidar.com. (2007).
- [24] Zhi Quan Zhou, Dave Towey, Pak-Lok Poon, and T. H. Tse. 2018. Introduction to the Special Issue on Test Oracles. *Journal of Systems and Software* 136 (2018), 187–187. <https://doi.org/10.1016/j.jss.2017.08.031> Editorial.
- [25] Zhi Quan Zhou, Shaowen Xiang, and Tsong Yueh Chen. 2016. Metamorphic testing for software quality assessment: A study of search engines. *IEEE Transactions on Software Engineering* 42, 3 (2016), 264–284.

Zhi Quan Zhou (zhiquan@uow.edu.au) is an associate professor in software engineering with the School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia.

Liquan Sun (ls168@uowmail.edu.au) is working toward the MPhil degree in computer science in the University of Wollongong, Australia. He is a software engineer with Itree, Australia.