DOI:10.1145/3241979

Metamorphic testing can test untestable software, detecting fatal errors in autonomous vehicles' onboard computer systems.

BY ZHI QUAN ZHOU AND LIQUN SUN

Metamorphic Testing of Driverless Cars

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.

Velodyne, the supplier of the Uber vehicle's LiDAR technology, said, "Our LiDAR is capable of clearly imaging Elaine and her bicycle in this situation. However, our LiDAR does not 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 "Our LiDAR can see perfectly well in the dark, as well as

» key insights

- Many software systems (such as AI systems and those that control self-driving vehicles) are difficult to test using conventional approaches and are known as "untestable software."
- Metamorphic testing can test untestable software in a very cost-effective way, using a perspective not previously used by conventional approaches.
- We detected fatal software faults in the LiDAR obstacle-perception module of self-driving cars and reported the alarming results eight days before Uber's deadly crash in Tempe, AZ, in March 2018.

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]

Question Concerning Every Human Life

Regardless of investigation outcomes, this Uber fatal accident raised a serious question about the perception capability of self-driving cars: Are there situations where a driverless car's onboard 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 obstacle in the roadway? This question is not specific to Uber cars but is general enough to cover all types of autonomous vehicles, and the answer concerns every human life. Unfortunately, our conclusion is affirmative. Even though we could not access the Uber system, we have managed to test Baidu Apollo, a well-known real-world self-driving software system controlling many autonomous vehicles on the road today (http://apollo.auto). Using a novel metamorphic testing method, we have detected critical software errors that could cause the Apollo perception module to misinterpret 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 as issue #3341 (https://github.com/ApolloAuto/apollo/issues/3341). We did not receive a response from Baidu until 10:25 P.M., March 19, 2018, MST—24 hours after the Uber accident. In its reply, the Apollo perception team confirmed the error. Before presenting further details of our findings, we first discuss the challenges of testing complex computer systems, with a focus on software testing for autonomous vehicles.

Testing Challenge

Testing is a major approach to software quality assurance. Deploying inadequately tested software can have serious consequences.[22] Software testing is, however, fundamentally challenged by the "oracle problem." An oracle is a mechanism testers use to determine whether the outcomes of test-case executions are correct.[2,24] Most software-testing techniques assume an oracle exists. However, this assumption does not always hold when testing complex applications. This is thus the oracle problem, a situation where an oracle is unavailable or too expensive to be applied. For example, when a software engineer is testing a compiler, determining the equivalence between the source code and the compiler-generated object program is difficult. When testing a Web search engine, the tester finds it very difficult 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 done without test automation. For testing self-driving vehicles, constructing a fully automated test oracle is especially difficult. Although in some situations the human tester can serve as an oracle to verify the vehicle's behavior, manual monitoring is expensive and error-prone. In the Uber accident, for example, the safety driver performed no safety monitoring, and because "humans monitoring an automated system are likely to become bored and disengaged," such testing is "particularly dangerous."[12]

The oracle problem is also reflected in the difficulty of 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's decision making.[21] Even for highly qualified human testers with full system specifications, it can still be difficult or even impossible to determine the correctness of every behavior of an autonomous vehicle. For example, in a complex road network, it is difficult for the tester to decide whether the driving route selected by the autonomous car is optimal.[3] Likewise, it is not easy to verify whether the software system has correctly interpreted the huge amount of point-cloud data sent from a LiDAR sensor, normally at a rate of more than one million data points per second.

"Negative testing" is even more challenging. While positive testing focuses on ensuring a program does what it is supposed to do for normal input, negative testing serves to ensure the program does not do what it is *not* supposed to do when the input is unexpected, normally involving 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 attempted by the development organization), how can the tester identify the conditions under which the vehicle could potentially do something wrong, as in, say, unintentionally striking a pedestrian? To a certain degree, tools called "fuzzers" could help 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 verification of the fuzz test results (outputs for millions of random inputs) extremely difficult, if not impossible.[5] In fuzzing, the tester thus looks only for software crashes (such as aborts and hangs). This limitation means not only huge numbers of test cases might need to be run before a crash but also 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.

Metamorphic Testing

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 the 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 can thus 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) and 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 the tester entered a search criterion C_1 and 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 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 AND C_2" is a new test case that can be constructed automatically 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 verified automatically through a testing program.

A growing body of research from both industry and academia has examined the MT concept and proved it highly effective.[4,7–9,13,18–20] The increasing interest in MT is not only due to it being able to address the oracle problem and automate test generation but also the perspective of MT has seldom been used in previous testing strategies and, as a result, 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]

MT for Testing Autonomous Machinery

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

For example, researchers from the Fraunhofer Center for Experimental Software Engineering in College Park, MD, developed a simulated environment in which the control software of autonomous drones was tested using MT.[14] The 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. The Fraunhofer researchers looked for behavioral differences of the drone when it was flying under these different (supposedly equivalent) scenarios. Their MRs required the drone should have consistent behavior, while finding 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, making the algorithm sensitive to certain numerical values and hence misbehavior under certain conditions, causing the drone to crash. The researchers 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 the researchers rotated all the objects in the environment, but not the sun, unexpectedly causing a shadow to fall on the landing pad in some orientations, revealing issues in the drone's vision system. The researchers solved this with a more robust vision sensor that was less sensitive to lighting changes.

Researchers from the University of Virginia and from Columbia University tested three different deep neural network (DNN) models for autonomous driving.[21] The inputs to the models were pictures from a camera, and the outputs were steering angles. To verify the correctness of the outputs, the researchers used a set of MRs based on

Figure 1. Input pictures used in a metamorphic test revealing inconsistent and erroneous behavior of a DNN (https://deeplearningtest.github.io/deepTest).[21]

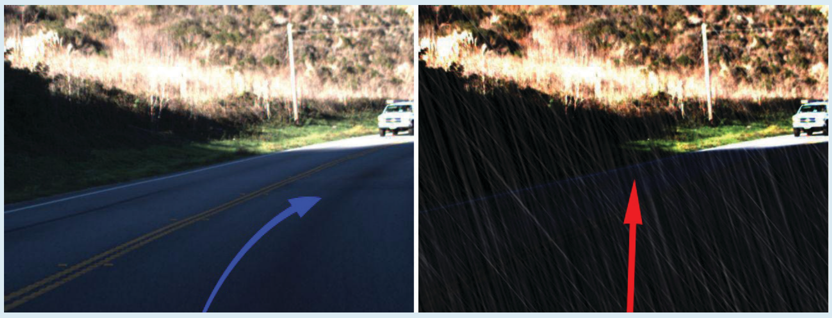


Figure 2. MT detected a real-life bug in Google Maps;[3,20] the origin and destination of the route were almost at the same point, but Google Maps generated an "optimal" route of 4.9 miles.

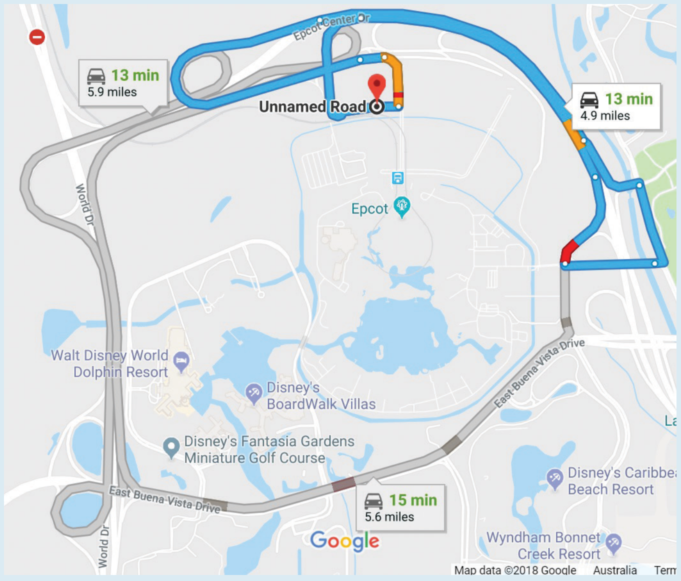

13 min
5.9 miles
13 min
4.9 miles
15 min
5.6 miles
Unnamed Road
Epcot Center Dr
World Dr
Epcot
Walt Disney World Dolphin Resort
Disney's BoardWalk Villas
Disney's Fantasia Gardens Miniature Golf Course
East Buena Vista Drive
Disney's Caribbean Beach Resort
Wyndham Bonnet Creek Resort
Google
Map data ©2018 Google Australia Terms

Figure 3. Results of experiments by category, visualized as 100% stacked column charts.

Each vertical column represents the comparisons of 1,000 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 1,000 pairs).

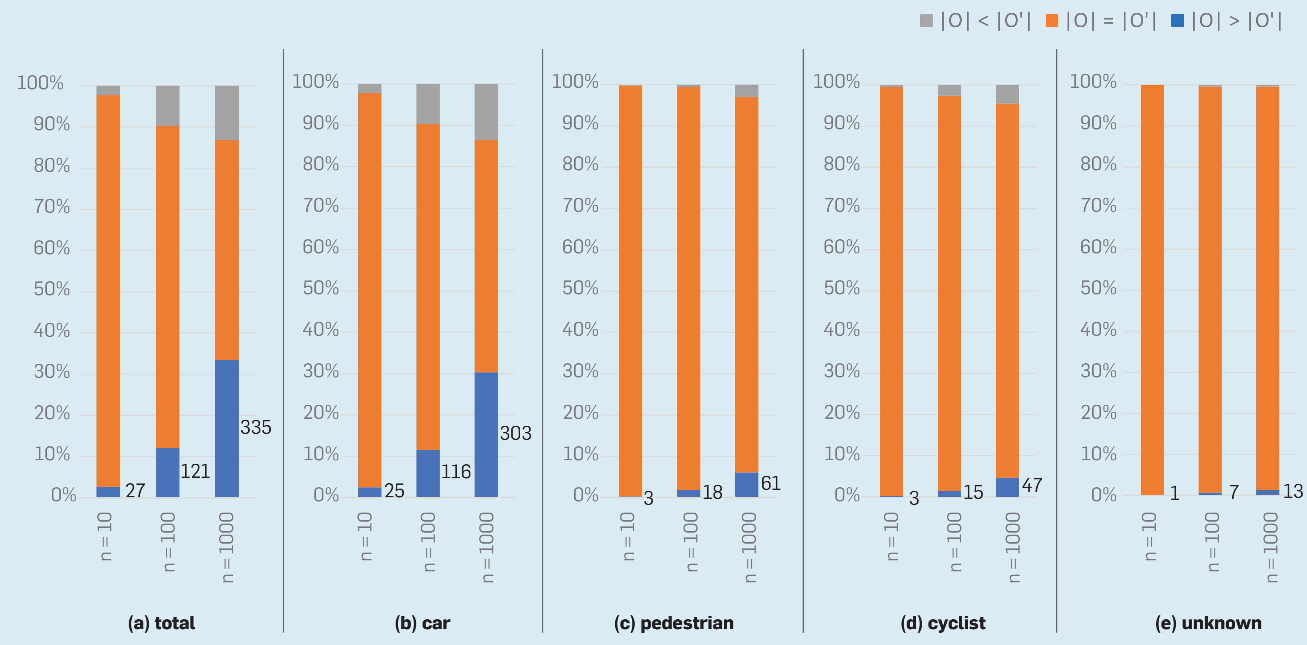


image transformations. The MRs said the car should behave similarly under variations of the same input (such as the same scene under different lighting conditions). Using these MRs, they generated realistic synthetic images based on seed images. These synthetic images mimic real-world phenomena (such 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 is an example, whereby 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.

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

LiDAR-Data-Interpretation Errors

The scope of the study conducted by the researchers from the University of Virginia and from Columbia University[21] was limited to DNN models. A DNN model is only part of the perception module of a self-driving car's software system. Moreover, while a DNN can take input from different sensors (such as a camera and LiDAR), they studied only the ordinary two-dimensional 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 tested was not real-life systems controlling autonomous cars but rather deep learning models that "won top positions in the Udacity self-driving challenge."

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

Software under test. More specifically, we tested Apollo's perception module (http://apollo.auto/platform/perception.html), which has two key components: "three-dimensional obstacle perception" and "traffic-light perception." We tested the three-dimensional obstacle perception component, which itself 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 tested

Test results; for each value of *n*, we compared 1,000 pairs of results.

number of added points (n)	|O| > |O'|	|O| = |O'|	|O| < |O'|
10	27	951	22
100	121	781	98
1,000	335	533	132

only the first, LiDAR obstacle perception (or LOP), which takes the three-dimensional point cloud data as input, as generated by Velodyne's HDL64E LiDAR sensor.

LOP resolves the raw point-cloud data using the following pipeline, as excerpted from the Apollo website (https://github.com/ApolloAuto/apollo/blob/master/docs/specs/3d_obstacle_perception.md):

HDMap region of interest filter (tested in our experiments). The region of interest (ROI) specifies the drivable area, including road surfaces and junctions that are retrieved from a high-resolution (HD) map. The HDMap ROI filter processes LiDAR points that are outside the ROI, removing background objects (such as buildings and trees along the road). What remains is the point cloud in the ROI for subsequent processing;

Convolutional neural networks segmentation (tested in our experiments). After identifying the surrounding environment using the HDMap ROI filter, the Apollo software obtains the filtered point cloud that includes only the points inside the ROI—the drivable road and junction areas. Most of the background obstacles (such as buildings and trees along the road) have been removed, and the point cloud inside the ROI is fed into the "segmentation" module. This process detects and segments out foreground obstacles (such as cars, trucks, bicycles, and pedestrians). Apollo uses a deep convolutional neural network (CNN) for accurate obstacle detection and segmentation. The output of this process is a set of objects corresponding to obstacles in the ROI;

MinBox builder (tested in our experiments). This object builder component establishes a bounding box for the detected obstacles;

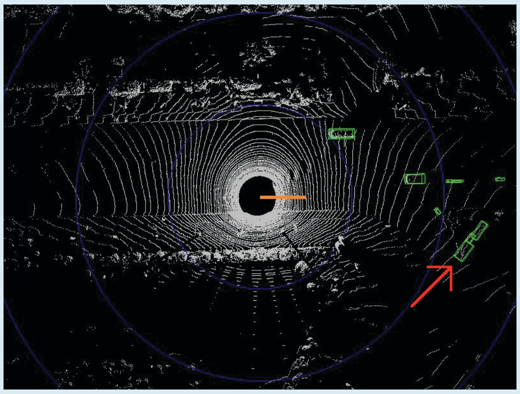
HM object tracker (not tested in our experiments). This tracker is designed to track obstacles detected in the segmentation step; and

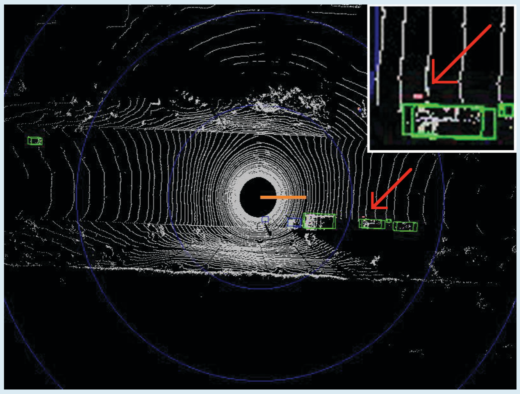
Sequential type fusion (not tested in our experiments). To smooth the obstacle type and reduce the type switch over the entire trajectory, Apollo uses a sequential type fusion algorithm.

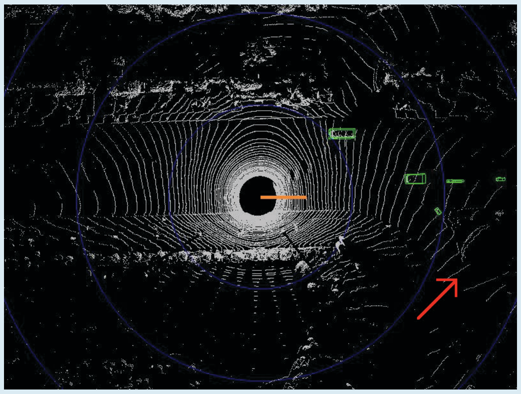
Our software-testing experiments involved the first, second, and third but not the fourth and fifth features, because the first three are the most critical and fundamental.

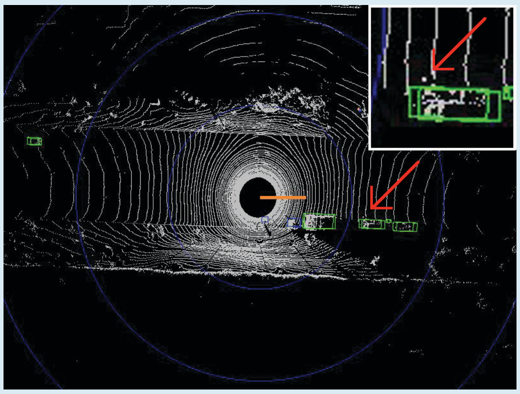
Our testing method: MT in combination with fuzzing. Based on the Baidu specification of the HDMap ROI filter, we identified the following meta-

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


(a) Original: 101,676 LiDAR data points; the green boxes were generated by the Apollo system to represent the detected cars.


(c) Original: 104,251 LiDAR data points; the small pink mark was generated by the Apollo system to represent a detected pedestrian.


(b) After adding 1,000 random data points outside the ROI, the three cars inside the ROI could no longer be detected.


(d) After adding only 10 random data points outside the ROI, the pedestrian inside the ROI could no longer be detected.

morphic relation, whereby the software under test is the LiDAR obstacle perception (LOP) subsystem of Apollo, A and A' represent two inputs to LOP, and O and O' represent LOP's outputs for A and A', respectively.

***MR*₁**. Let A and A' be two frames of three-dimensional 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. Also let O and O' be the sets of obstacles identified by LOP for A and A', respectively (LOP identifies only obstacles within the ROI). The following relation must then hold: $O \subseteq O'$.

In MR_1, the additional LiDAR data points in A' could represent small particles in the air or just some noise from the sensor, whose existence is possible.[23] MR_1 says the existence of some particles, or some noise points, or their combination, in the air far from 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 interfere with the detection of a pedestrian in front of the vehicle. This requirement is intuitively valid and agrees with the Baidu specification of its HDMap ROI filter. According to the user manual for the HDL64E LiDAR sensor, it can be mounted atop the vehicle, delivering a 360° horizontal field of view and a 26.8° vertical field of view, capturing a point cloud with a range up to 120 meters.

We next 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"—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 on the morning of September 6, 2017. In each series of experiments, we first randomly extracted 1,000 frames of the point cloud data; we call each such frame a "source test case." For each source test case t, we ran the LOP software to identify its ROI and 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 three-dimensional space outside the ROI of t; we determined the value of the z coordinate of each point 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. We then ran the LOP software for t', producing O', the set of detected obstacles. Finally, we compared O and O'. We conducted three series of experiments: for $n = 10$, 100, and 1,000. We thus ran the LOP software for a total of $(1{,}000 + 1{,}000) \times 3 = 6{,}000$ times, processing 3,000 source test cases and 3,000 follow-up test cases.

As an extreme example, a small insect 100 meters away—outside the ROI—should not interfere with the detection of a pedestrian in front of the vehicle.

Test results. In our experiments, for ease of implementation of MR_1, we did not check the subset relation $O \subseteq O'$ but instead 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. That is, if $|O| > |O'|$, then there must be something wrong, as one or more objects in O must be missing in O'.

The results of our experiments were quite surprising; the table here summarizes the overall results. The violation rates (that is, cases for $|O| > |O'|$ out of 1,000 pairs of outputs) were 2.7% (= 27 ÷ 1,000), 12.1% (=121÷ 1,000), and 33.5% (= 335 ÷ 1,000), for n = 10, 100, and 1,000, respectively. This means as few as 10 sheer random points scattered in the vast three-dimensional space outside the ROI could cause the driverless car to fail to detect an obstacle on the roadway, with 2.7% probability. When the number of random points increased to 1,000, the probability became as high as 33.5%. According to the HDL64E user manual, the LiDAR sensor generates more than one million data points per second, and each frame of point cloud data used in our experiments normally contained more than 100,000 data points. The random points we added to the point cloud frames were thus trivial.

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

Each vertical column in Figure 3 includes a subsection in blue, corresponding to MR_1 violations. They are labeled with the actual numbers of $|O| > |O'|$ cases. We observed that all these numbers were 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 was greatest, followed by "pedestrian," "cyclist," and "unknown."

Figure 4a and Figure 4b show a real-world example revealed by our test, whereby three cars inside the ROI could not be detected after we added 1,000 random points outside the ROI. Figure 4c and Figure 4d show another example, whereby a pedestrian inside the ROI (the Apollo system depicted this pedestrian with the small pink mark in Figure 4c) could not be detected after we added only 10 random points outside the ROI; as shown in Figure 4d, the small pink mark was missing. As mentioned earlier, we reported the bug to the Baidu Apollo self-driving car team on March 10, 2018. On March 19, 2018, the Apollo team confirmed the error by acknowledging "It might happen" and suggested "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. Our failure-causing metamorphic test cases (those with the random points) could thus serve this purpose.

Conclusion

The 2018 Uber fatal crash in Tempe, AZ, revealed the inadequacy of conventional testing approaches for mission-critical autonomous systems. We have shown MT can help address this limitation and enable automatic detection of fatal errors in self-driving vehicles that operate on either conventional algorithms or deep learning models. We have introduced an innovative testing strategy that combines MT with fuzzing, reporting how we used it to detect previously unknown fatal errors in the real-life LiDAR obstacle perception system of Baidu's Apollo self-driving software.

The scope of our study was limited to LiDAR obstacle perception. Apart from LiDAR, an autonomous vehicle 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." Moreover, there can also be cameras, which are particularly useful for detecting visual features (such as the color of traffic lights). Our testing technique can be 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, combined with existing verification and validation methods, to make driverless vehicles safer.

Acknowledgments

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

References

1. Baidu, Inc. *Apollo Reference Hardware*, Mar. 2018; http://apollo.auto/platform/hardware.html
2. Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering 41*, 5 (May 2015), 507–525.
3. Brown, J., Zhou, Z.Q., and Chow, Y.-W. Metamorphic testing of navigation software: A pilot study with Google Maps. In *Proceedings of the 51st Annual Hawaii International Conference on System Sciences* (Big Island, HI, Jan. 3–6, 2018) 5687–5696; http://hdl.handle.net/10125/50602
4. Chen, T.Y., Kuo, F.-C., Liu, H., Poon, P.-L., Towey, D., Tse, T.H., and Zhou, Z.Q. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys 51*, 1 (Jan. 2018), 4:1–4:27.
5. Chen, T.Y., Kuo, F.-C., Ma, W., Susilo, W., Towey, D., Voas, J., and Zhou, Z.Q. Metamorphic testing for cybersecurity. *Computer 49*, 6 (June 2016), 48–55.
6. Chen, T.Y., Tse, T.H., and Zhou, Z.Q. Fault-based testing without the need of oracles. *Information and Software Technology 45*, 1 (2003), 1–9.
7. Donaldson, A.F., Evrard, H., Lascu, A., and Thomson, P. Automated testing of graphics shader compilers. *Proceedings of the ACM on Programming Languages 1* (2017), 93:1–93:29.
8. Jarman, D.C., Zhou, Z.Q., and Chen, T.Y. Metamorphic testing for Adobe data analytics software. In *Proceedings of the IEEE/ACM Second International Workshop on Metamorphic Testing,* in conjunction with the *39th International Conference on Software Engineering* (Buenos Aires, Argentina, May 22). IEEE, 2017. 21–27; https://doi.org/10.1109/MET.2017.1
9. Kanewala, U., Pullum, L.L., Segura, S., Towey, D., and Zhou, Z.Q. Message from the workshop chairs. In *Proceedings of the IEEE/ACM First International Workshop on Metamorphic Testing,* in conjunction with the *38th International Conference on Software Engineering* (Austin, TX, May 16). ACM Press, New York, 2016.
10. Le, V., Afshari, M., and Su, Z. Compiler validation via equivalence modulo inputs. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, U.K., June 9–11). ACM Press, New York, 2014, 216–226.
11. Lee, D. Sensor firm Velodyne 'baffled' by Uber self-driving death. *BBC News* (Mar. 23, 2018); http://www.bbc.com/news/technology-43523286
12. Levin, S. Uber crash shows 'catastrophic failure' of self-driving technology, experts say. *The Guardian* (Mar. 23, 2018); https://www.theguardian.com/technology/2018/mar/22/self-driving-car-uber-death-woman-failure-fatal-crash-arizona
13. Lindvall, M., Ganesan, D., Árdal, R., and Wiegand, R.E. Metamorphic model-based testing applied on NASA DAT — An experience report. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering* (Firenze, Italy, May 16-24). IEEE, 2015, 129–138.
14. Lindvall, M., Porter, A., Magnusson, G., and Schulze, C. Metamorphic model-based testing of autonomous systems. In *Proceedings of the Second IEEE/ACM International Workshop on Metamorphic Testing,* in conjunction with the *39th International Conference on Software Engineering* (Buenos Aires, Argentina, May 22). IEEE, 2017, 35–41.
15. Ohnsman, A. LiDAR maker Velodyne 'baffled' by self-driving Uber's failure to avoid pedestrian. *Forbes* (Mar. 23, 2018); https://www.forbes.com/sites/alanohnsman/2018/03/23/lidar-maker-velodyne-baffled-by-self-driving-ubers-failure-to-avoid-pedestrian
16. Posky, M. LiDAR supplier defends hardware, blames Uber for fatal crash. *The Truth About Cars* (Mar. 23, 2018); http://www.thetruthaboutcars.com/2018/03/lidar-supplier-blames-uber/
17. Regehr, J. *Finding Compiler Bugs by Removing Dead Code.* Blog, June 20, 2014; http://blog.regehr.org/archives/1161
18. Segura, S., Fraser, G., Sanchez, A.B., and Ruiz-Cortés, A. A survey on metamorphic testing. *IEEE Transactions on Software Engineering 42*, 9 (Sept. 2016), 805–824.
19. Segura, S. and Zhou, Z.Q. Metamorphic testing: Introduction and applications. ACM SIGSOFT webinar, Sept. 27, 2017; https://event.on24.com/wcc/r/1451736/8B5B5925E82FC9807CF83C84834A6F3D
20. Segura, S. and Zhou, Z.Q. Metamorphic testing 20 years later: A hands-on introduction. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering* (Gothenburg, Sweden, May 27–June 3, 2018). ACM Press, New York, 2018.
21. Tian, Y., Pei, K., Jana, S., and Ray, B. DeepTest: Automated testing of deep neural network-driven autonomous cars. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering* (Gothenburg, Sweden, May 27–June 3, 2018). ACM Press, New York, 2018.
22. Vassilev, A. and Celi, C. Avoiding cyberspace catastrophes through smarter testing. *Computer 47*, 10 (Oct. 2014), 102–106.
23. Velodyne, *Velodyne's HDL-64E: A High-Definition LiDAR Sensor for 3-D Applications*, White Paper, 2007; https://www.velodynelidar.com/
24. Zhou, Z.Q., Towey, D., Poon, P.-L., and Tse, T.H. Introduction to the special issue on test oracles. *Journal of Systems and Software 136* (Feb. 2018), 187; https://doi.org/10.1016/j.jss.2017.08.031
25. Zhou, Z.Q., Xiang, S., and Chen, T.Y. Metamorphic testing for software quality assessment: A study of search engines. *IEEE Transactions on Software Engineering 42*, 3 (Mar. 2016), 264–284.

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

Liqun Sun (ls168@uowmail.edu.au) is pursuing an M.Phil. degree in computer science at the University of Wollongong, Wollongong, NSW, Australia, and a software engineer at Itree, Wollongong, Australia.