Attacks on Autonomous Driving Systems:Automated Model-based Detection

EECE 571K: Security and Reliability in IoT - Midterm report

Aarti Kashyap
Student number: 19115724
Electrical and Computer Engineering, UBC
akashyap18@ece.ubc.ca

Abdul Rehman Anwer
Student number: 50342690
Electrical and Computer Engineering, UBC
abanwer@ece.ubc.ca

***Abstract*—Autonomous vehicles (AV) have seen an immense increase in popularity recently and are expected to become fully operational on normal roads in near future. The safety critical nature of autonomous driving systems (ADS) of AVs makes quantifying the resilience properties of such systems highly important, for ensuring public and governmental support for their adoption. Resilience of ADS in general has been explored before but effect on reliability of ADS due to specific components such as sensors present in ADS still remains to be explored. Prior work in this area focuses on using Fault Injection without any systemic approach to assess the safety and reliability of autonomous vehicles which also test the sensors present in the system do not provide good test coverage due to diverse range of possible inputs for control systems of ADS. We propose using a systematic approach to find vulnerabilities in ADS introduced due to sensors and compare its performance overhead and coverage with fuzz testing techniques.**

I. Introduction

Fully automated self driving vehicles are poised to take the roads in near future. Partial self driving systems have already been deployed and tested for over a billion miles [4], providing valuable sensor data that is used to improve the performance of automated driving systems. The adoption of autonomous driving systems (ADS) on a large scale in real world is highly dependent on their reliability. Moreover, for the acceptance of deployment of such safety critical systems by wider public, makes it necessary to understand the resilience profiles of autonomous driving systems, identify and improve weaknesses in current designs and exhaustively test these fixes to increase public confidence.

Autonomous Driving System are composed of multiple sub-systems, the major parts being the sensor arrays and the control systems. A wide range of sensors like RGB cameras, radars, LIDARs, sonar, GPS etc. are used to provide information about the environment to the control system of ADS, which usually uses a Deep Neural Network to find the best course of action and gives commands to actuators, to control the movement of the vehicle. Such a highly integrated systems introduce various concerns about the overall reliability of the system. Moreover as ADS are highly safety critical, regulations and standards impose stringent reliability requirements on their components. For example the SDC FIT rate (Failure-in-Time rate) for of SoC used for DNN inferencing specified by ISO-26262 is 10 i.e. ten failures in one billion hours of operation [7]. These strict reliability requirements, demand that a reliability analysis be performed on integrated systems and vulnerabilities or faults that can propagate, from one sub-system to other sub-systems be identified and mitigation techniques applied to make the system resilient to such faults.

Studying the autonomous driving system is hindered by resource intensive nature of training and testing solutions in real world scenarios. Dosovitskiy et al. [3] recently introduced CARLA, which is an open source simulator for training and validating autonomous driving platforms. It provides an urban environment setting from which data is captured using a flexible array of sensors and signals that are normally present in ADS, this simulated data can then be used to train and test different sub-systems of an ADS. As CARLA also provides sensor reading in addition to images captured by camera, it can be used to study the effect of a vulnerability present in a single component (for example the LIDAR sensor) on overall reliability of the whole ADS. CARLA operates in a server-client model. Server simulates the environment and generates readings for different sensors which are then provided to the client which acts as a driving agent (DA) taking different control decisions based on the provided sensor inputs. Separation of server and client functionality permits implementation of a wide range of DAs.

As per our knowledge, we are the first to attempt to find the failures caused by sensors in a self-driving car.

Section 2 introduces the related work done in the resilience checking of self-driving cars and how our work is different from the existing literature. Section 3 introduces the background related to self-driving cars. This section talks about the different sensors which form the core components of the self-driving cars and then explains the in-depth working of LIDAR, since finding failures due to LIDAR is the core contribution of our work. Section 4 walks through the fault model, type of faults and the metrics we use in our study to evaluate autonomous driving systems. Section 5 briefly describes configuration of CARLA and driving agent we use

in our study and describes the method of Fault injection we use in our study and the limitations of our chosen methods and environment.

II. Related Work

Reliability and safety of individual components in ADS have been previously studied. Li et al. [7] studied the impact of soft errors on DNN using Fault injections on hardware accelerators and proposed solutions for making DNNs more resilient. Using Fault injections for testing reliability of system has inherent disadvantage that for complex systems like DNNs a huge number of FI are required to get statistically meaningful insights, moreover the coverage that FI provides is also dependent on the input used for FI, so in case of DNN where the input has a diverse range, performing FI to get meaningful resilience profile of a DNN sub-system of an ADS can be quite resource intensive, otherwise some corner case may be missed during the analysis. Keeping these things in mind Pi et al. [8] introduced DeepXplore, which tests deep learning systems in a systematic way by finding the inputs for DL systems, that will result in mismatch in output for different DL systems that provide same functionality. In this way corner cases that are not captured by regular testing and training cycle in neural networks are found thus exposing erroneous behavior of the system without a significant overhead of finding corner cases manually. DeepXplore also finds input that maximize the fraction of neurons activated when tested, thus increasing the tests coverage of DNNs. DeepXplore can provide a good defense against vulnerabilities and erroneous behaviors in DNN part of ADS but for overall reliability of the program other components of the system also need to be analyzed.

Dosovitskiy et al. [5] developed AVFI which is a fault injector that performs resilience assessment of complete autonomous vehicle(AV) systems. AVFI inject faults in without any systematic approach in AV system which consists of CARLA simulator and a driving agent which performs the control actions. AVFI can inject hardware faults (modeling soft errors), data faults (modeling error in sensor values), timing faults and machine learning faults (errors leading to wrong prediction). Jha et al. [6] developed Kayotee which is a FI based tool to inject faults in an ADS (similar to AVFI) and then classifying errors and safety violations impacting the reliability of autonomous vehicles. Kayotee injects faults in both hardware and software components of proprietary Nvidia DriveWorks platform, in a closed looped environment and then error propagation and masking characteristics of the ADS compute stack are evaluated using a predefined ontology model. Kayotee and AVFI provides a holistic view of the system but its fault coverage is also limited, as only those errors can be studied which have been simulated using FI. As whole system is taken into consideration during holistic analysis, trying to increase the test coverage using FI can lead to state space explosion.

III. Background

A. Autonomous Cars

Self-driving autonomous cars use various autonomous technologies to provide an effortless mode of transportation. In order for autonomous cars to work effectively a proper synchronization of advanced sensors collecting data from surrounding environments and advanced algorithms processing data and the vehicle in real-time.

The main components of a self-driving car are sensors (we will talk about them in more details in the next subsection), computer processors, batteries, back-up systems (eg. steering, braking etc.), rounded-shapes which maximizes the sensors coverage area and the interiors(designed for riding, not for driving).

B. Sensors

Sensors are the core components in a self-driving car. We will talk about them in a little more detail, since our research goal is to find error rate in self-driving cars caused by sensor faults.

The core sensors of a self-driving car are as follows:

1) Radar sensors: Radar sensors help the autonomous vehicles in determining the road dynamics such as detours, collisions, navigation by sending signals to the on-board processor. The on-board processor on receiving the signals from the radar sensors takes decisions such as applying brakes and/or move out of the way. The decisions made by the processor or the brain of the vehicle don't depend entirely on the radar sensors. The processor decision also depends on the gyroscopes and wheel encoders to send accurate and on time signals to it. The radar sensors are usually mounted on the bumper (two in the front and two in the back).

2) Optics: High powered cameras in conjunction with Radar sensors are another vital component of the autonomous vehicles. The camera setup on each car differs according to the manufacturer. The main goal of the camera placement is to get the surrounding views covered properly, so that the processor can take correct decisions. Some manufacturers use a single camera on the windshield [9] whereas some vehicles require multiple cameras placed equidistant from each other to get overlapped views which are sent to the processor. The processor uses the views to calculate the depth of the field and/or perform semantic segmentation.

3) GPS: Global Positioning Sensors is very similar to the navigation maps we use in our mobile phones. However, GPS in autonomous vehicles is very important since we enter the destination point and the start point of our trip. The GPS works in conjunction with LIDAR, Radar Sensors and Deep learning software [1] in order to guide the vehicle in the correct direction to reach the destination.

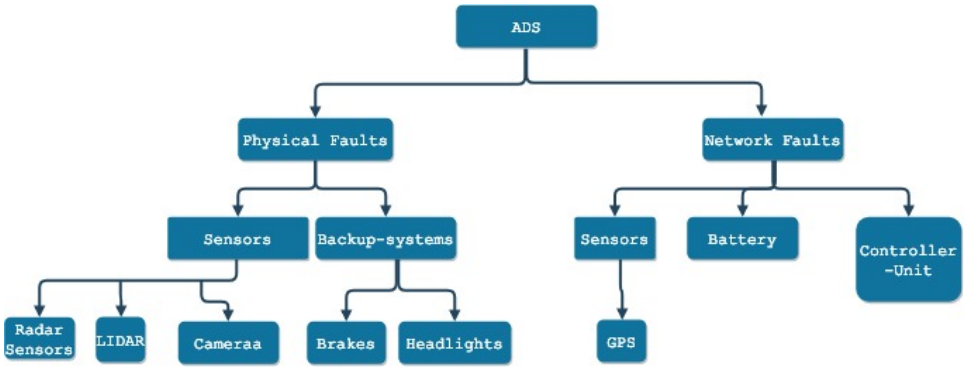

Fig. 1. This shows a general fault model of Autonomous Driving Systems. We have kept it generic enough to be considered for different Autonomous systems. However, our work is oriented towards autonomous cars.

4) LIDAR: Laser Illuminating Detection and Ranging is the sensor on which we are focusing on in this work. LIDAR unit resembles a spinning siren light and ranges up-to 100 meters. The goal of LIDAR is to provide driver-less cars with accurate long range detection. LIDAR continuously performs 360 degree rotation in order to scan the world around. The sensor generates raw information about the world and the information is sent to the processor which digs out the relevant information in order to guide the vehicle and take decisions.

C. LIDAR

Since we are trying to determine the failure rates of LIDAR, we give a brief overview of how LIDAR units work in an autonomous car.

The autonomous car LIDAR units follow four steps in order to function:

1) The LIDAR unit which is continuously rotating emits a laser beam generated by the laser emitter present inside LIDAR.
2) The laser bounces off the an object. The object is usually something solid and the moment, laser hits the first solid object it bounces off and returns to LIDAR.
3) The gathered light is then bounced downwards to the receiver, where the light is interpreted.
4) This is a continuous process in all directions and the data gets sent to the processor which generates a map of its surroundings.

IV. Approach

Our contribution is two-fold in this work. First, by an in depth understanding of autonomous vehicles, we device a fault model for the vehicle in general. We specifically focus on the sensor faults and design a fault-model specific to the sensors. Second, based on our fault model, we inject faults in order to obtain the failure rate of the system caused due to the sensor faults. We attempt fault injection in order to get maximum coverage of the system in an optimal manner.

A. Fault-model for ADS

Analyzing the system, we designed a fault model for autonomous driving systems. We categorized the fault occurrence in two ways: due to some physical damage or due to network issues. These two categories of faults cover almost all possible components of the systems which can lead to a failure of the system.

In Figure 1 we try to give a complete fault model of autonomous vehicles. However, covering the failures caused by each component is out of scope of our current paper. Also, analyzing failure rates in general instead of component wise failure rates will not provide us with useful set of results. Hence we built a complete fault-model of the system initially. After building a comprehensive fault-model, based on our judgment, we decided to focus on one of the core set of components of ADS.

The core component we selected were the sensors used in ADS. However, every sensor used in ADS is very detailed oriented and performs multiple functionalities. So out of the four major sensors which we have explained in further details in Section 3, we focused on LIDAR.

B. Fault-model of LIDAR

LIDAR is a very expensive sensor and hence finding the failure rates of the system, caused due to LIDAR can help the designers understand how to mitigate the faults such that they don't lead to failures. ADS being safety critical systems, can cause a lot of harm (death in the worst case scenario) in case of a failure.

We will be providing the fault-model for ADS in our final report.

C. Fault Injection

In this section we describe the faults that we inject in sensor data according to our fault model and the effects of these faults. In our study we focus on two type of faults.

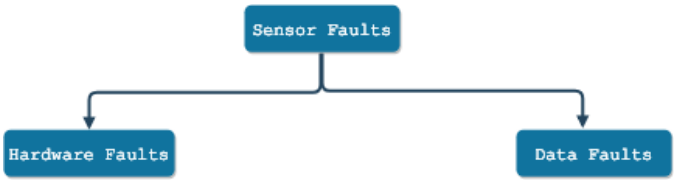

Fig. 2. In the sensor faults (LIDAR), we consider hardware and data faults.

1) Hardware faults: These faults represent the soft errors that can occur in the data that is recorded by the sensors. These faults can occur in the registers of sensor units, in communication links between sensor and control unit and with in registers where these values are stored in the main ADS controller. Regardless of the location where the fault occurs, we assume that it is always activated and observed by the client when it reads the sensor data. This type of faults in real life can introduce single and multiple bit flips randomly in the sensor readings, which is modeled by our FI technique.

2) Data Faults: These faults refers to the condition in which the sensors give a readings which are not correct according to the environment. These faults can occur due to manufacturing defects in the sensors and various environmental conditions. Environmental conditions can refer to weather and other characteristics of the surroundings that are used in decision making process by the ADS (e.g. lane markings on road). For example, A LIDAR mapping its surrounded by laser can be effected by rainy weather such that the laser bouncing from the surrounding objects is effected by rainy weather thus the mapping produced by LIDAR is not representative of real environment.

In our study with random Fault injection we study each fault separately, so for a set of related experiments we only inject one type of fault. Only one fault is injected in each execution run and its effect on the overall operation is observed. The goal and path to navigate is constant in each trial so that outcome of each trail can be compared with output of a golden run, in which no fault was injected. We use the following metrics to quantify the resilience of ADS which are inline with previous work in this domain [5].

1) Success percentage: This metric represents the fraction of total trials in which the ADS was successful in achieving the specified target when faults are injected.

2) Violations per trial: This metric captures the traffic violations that occur during each trial. The violations that we consider for calculating this metric are lane violation and curb violations (reasons for not considering collisions with dynamic objects are explained in V-A).

The two metrics used provide and opportunity to study the effects of faults on the overall functioning of AV as well as the lower level deviations from normal behavior, which may not lead to outright failure but violate safety properties.

V. Research Implementation

In this section we describe the architecture of of CARLA in detail and the present our fault injection methodology.

A. CARLA

CARLA is an open source simulator built on Unreal Engine 4. CARLA provides simulation of realistic physics, NPC logic and dynamic world. CARLA simulates actions of an autonomous vehicle (AV) and the environment that this AV experiences, with ability to control the actions of the AV by an external driving agent, that decides actions to be taken by AV, based on the data that it retrieves from simulation engine. Figure 3 shows the high level architecture of CARLA. CARLA operates in a server-client model, with functionality between these divided in following manner.

1) Client: It runs the simulation and renders the environment that the AV experiences.

2) Server: It controls the parameters of the simulation and also controls the actions of the AV based on predefined algorithms.

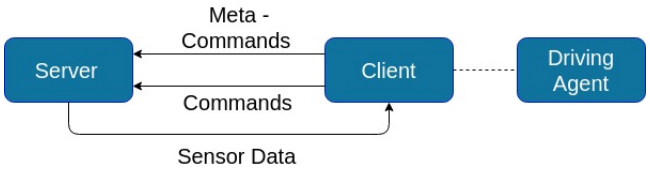

Fig. 3. High level architecture of CARLA, Server sends sensor data to the client which integrates with driving agent and returns driving commands to server.

Due to this client-server architecture CARLA provides flexibility to swap multiple driving algorithms (called driving agents in CARLA terminology) and driving environments easily. The server provides the ability to render a range of environments by changing the weather, number of vehicles and pedestrians etc in the simulation environment. The server also provides the output of various sensors attached to AV to the driving agent. The client connects to the server using sockets and provides two type of inputs, commands and meta-commands, to the server. Commands control the behavior of the vehicles i.e braking, steering, accelerating etc. and are provided by the driving agent which is integrated with client side in CARLA using python APIs. Meta-commands are used to control the behavior of simulation.

Although CARLA provides ability to simulate complex environments, Dosovitskiy et al. [3] have shown that for a range of driving agents, only simple paths with no dynamic actors present produce successful results for more than 90 percent trials. Following this information we primarily select simple environmental conditions and objectives for our experiments.

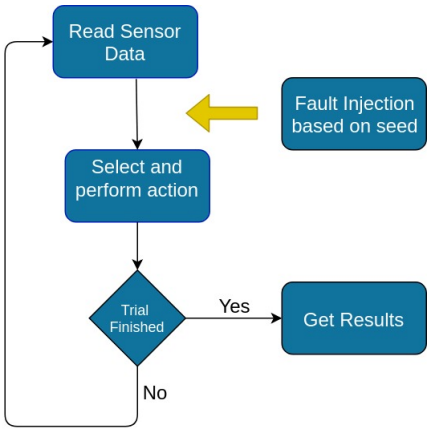

Fig. 4. Life cycle of an experiment trial on client side, Fault is injected by altering sensor readings received by client.

B. Driving Agent

As mentioned in the previous section the behavior of the vehicle in te simulation is controlled by the driving agent. As we want to study the resilience of CARLA, we have chosen an already developed driving agent, which uses conditional imitation learning to train a deep neural network [2], which after offline training, is used for inferencing for taking decision during simulation. We take an already trained model and use that in our Fault injection phase. Dosovitskiy et al. [3] have shown that success rate of the trained driving agent for the chosen environmental conditions is greater than 95%.

C. Fault Injection method

As mentioned in V-A CARLA provides a client server architecture. We exploit this boundary between client and server for our fault injection experiments.

Figure 4 provides an overview of method of fault injection used in one trial. The CARLA platform provides an ability to run multiple trails of the same experiments (same environment and goals). Each trial involves multiple steps. In each step data is read from the sensors, this data is provided to the controller which based on some predefined algorithm and current sensor readings, gives command to the AV, after the command its checked if the objective of the trial is achieved or the timeout has happened. If any of these condition exists we end the current trial and the using the measurements from the server, the metrics defined in IV-C are calculated for this trial and after that we move on to the next trial.

The sensor readings are provided in form of dictionary to driving agent from the client, which can be easily accessed in the code. We insert a code block in between reading data from server and sending it to driving agent. This code block changes the reading of the sensor under study based on the fault that we are testing. This code block injects fault in only one step during each trial, as we intend to study the effect of individual faults.

The fault space for even one sensor using approach mentioned above is quite big, which may lead to state space explosion. We intend to use this approach as a stepping stone towards finding a systematic approach for FI.

D. Environment Setup

We consider the following two environment setups for performing fault injections.

1) Straight Path: The ideal path of the AV is a straight line and the distance between starting and ending position is 200m.

2) Static Environment: No dynamic objects (i.e. NPC vehicles, pedestrians etc.) are present in the simulation environment.

References

[1] NVIDIA BLOG. Beyond gps: How hd maps will show self-driving cars the way. https://blogs.nvidia.com/blog/2016/04/05/self-driving-cars-2/.

[2] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[4] Tesla Inc. Vehicle safety report (Q4-2018). https://www.tesla.com/en_CA/VehicleSafetyReport.

[5] Saurabh Jha, Subho S. Banerjee, James Cyriac, Zbigniew T. Kalbarczyk, and Ravishankar Iyer. Avfi: Fault injection for autonomous vehicles. pages 55–56, 06 2018.

[6] Saurabh Jha, Timothy Tsai, Siva Hari, Michael Sullivan, Zbigniew Kalbarczyk, Stephen W. Keckler, and Ravishankar K. Iyer. Kayotee: A Fault Injection-based System to Assess the Safety and Reliability of Autonomous Vehicles to Faults and Errors. In *International Workshop on Automotive Reliability and Test (ART), 2018*. IEEE, 2018.

[7] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Jeol Emer, and Stephen Keckler. Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), 2017*. ACM, 2017.

[8] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Symposium on Operating Systems Principles (SOSP), 2017*, pages 1–18. ACM, 2017.

[9] MIT Technology Review. One camera is all this self-driving car needs. https://www.technologyreview.com/s/539841/one-camera-is-all-this-self-driving-car-needs/.