Achieving Crash Fault Tolerance in Autonomous Vehicle Autopilot Software Stacks Through Safety-Critical Module Rejuvenation

Federico Lucchetti, Marcus Voelp

Critical and Extreme Security and Dependability Group (CritiX), Interdisciplinary Centre for Security Reliability and Trust, University of Luxembourg, Luxembourg; email: {federico.lucchetti,marcus.voelp}@uni.lu

Abstract

Autonomous driving vehicles (ADV), have been in recent years, victims of their own success. Through their use of increasingly sophisticated sensor modalities and deep learning capabilities, ADVs have not only learned how to probe their chaotic environment with higher granularity coupled with smooth trajectory execution but also inherited all the vulnerabilities that were hiding behind these new features. Ensuring the safety of ADVs is crucial since a simple fault along their underlying autopilot software stack can lead to catastrophic accidents with the loss of human lives. Therefore we propose a crash-fault tolerant scheme that can be triggered whenever a crash fault of the safety critical submodules of the autopilot software stack is detected, which executes an emergency trajectory and effectively steers the car into a safe spot where the autopilot can be rejuvenated. We implement and evaluate the efficacy of this recovery scheme in the Apollo ADV software stack in conjunction with the SVL simulator. Keywords: autonomous driving vehicles, rejuvenation, crash fault tolerance, simplex architecture, apollo software stack.

1 Introduction

Autonomous driving vehicles (ADVs) have become more advanced in recent years, incorporating advanced features such as deep neural networks and intricate obstacle prediction modules. These developments are crucial to the success of computer vision algorithms, precise ADV trajectory planning, and smooth control algorithms, and have raised the level of driving automation to new heights. However, this increasing sophistication also comes with a downside. The growing complexity of ADVs increases their vulnerability to malicious intrusions and introduces new potential faults, which can lead to dangerous and even deadly outcomes.

Another limitation is that the autonomous driving software stack is typically regarded as reliable and expected to withstand accidental failures and cyber-attacks, which lately have become increasingly more advances. This assumption places an additional constraint on the development of autonomous driving technology, as it must be designed to be highly robust and resistant to potential security breaches.

Indeed, ADVs have been involved in numerous tragic incidents over the past two decades. One major contributing factor to these accidents has been unintended accelerations (UA), which have resulted in the deaths of 89 individuals [1]. Such behaviors can have two possible origins: they may result from an accidental internal fault or the absence of a fail-safe mechanism, or they may be intentionally provoked by a malicious attacker [2].

The adoption of ADVs on a large scale depends on convincing human drivers of their reliability. Therefore, the resilience of ADVs must play a critical role in ensuring their effective adoption by the general public. It is inevitable that faults will occur at any level, so it is crucial to equip ADVs with mechanisms that enable them to tolerate these faults. In the event of faults, a responsibility gap arises, in which it is unclear who can be held responsible for any unintended catastrophic outcomes. This gray zone is even wider due to the over-reliance of modern ADV modules on artificial neural networks (NN) whose safety and reliability properties are seldom studied in conjunction with the entire ADV software stack [3].

In addition to their black-box nature, NNs are susceptible to common faults that can originate from either software or hardware issues [4]. In the former, malicious intruders can reprogram, evade, or data-poison NNs during either the inference or training phase [5, 6, 7]. In the latter case NNs can for instance be targets of single-event upsets leading to permanent or transitory faults such as stuck-at or bit-flip types which can alter the parameter space of the NN or cause an erroneous computation of hidden layers' activation functions [8].

Similarly, sensors are not immune to attacks. Malicious actors can modify the lane-keeping system by installing dirty road patches, causing the ADV to drift out of its lane [9]. Jamming the camera modules or executing LIDAR spoofing attacks in order to inject false obstacle depth can lead to false sensor data and cause the ADV's data processing chain to compute erroneous control commands [10, 11]. In these cases, the health of the sensors remains uncompromised, and traditional fault detection schemes are unable to detect the attacks.

Autonomous driving software stacks, such as Apollo Baidu, typically consist of a series of interconnected modules that

process information sequentially in a event-triggered manner. Sensor values are gathered and processed in the prediction module, then forwarded to the prediction module for obstacle trajectory prediction. A planning module computes the safest and shortest trajectory given the constraints forwarded by the prediction module. The ADV trajectory coordinates are translates into control commands by the control module and sent to the electronic control unit (ECU) for actuation. Due to the downstream interdependence of these modules and the causal interlinking of the computation and safe execution of control commands, the failure of an intermediary module can propagate throughout the information processing chain and result in unforeseen behaviors.

To mitigate the risk of single points of failure in ADV software, efforts have been made to employ redundancy by gathering data from multiple sources, such as RGB cameras, LIDAR, and RADAR, and fusing it to minimize the impact of a faulty device [12]. However, redundancy comes at an additional computational cost, and certain modules, such as GPU-resource greedy NNs, cannot be easily replicated. Other have developed adaptive control algorithms and equipped sensor fusion modules with fault detection capabilities to enhance the overall fault tolerance of ADVs [13]. However, validating ADV software in a real physical environment is costly and not scalable for all possible driving scenarios. Therefore, interfacing physics simulators like SVL [14] with ADV software stacks is crucial to ensure quality assurance in the automotive sector as required by the evolving standard ISO 21448: Safety of the Intended functionality [15].

1.1 Related Work

The studies referenced in [12] and [13] investigated fault tolerance in ADV systems. The former focused on fusing data from different sensor modalities to mask potentially faulty outputs, while the latter developed a model adaptive control algorithm with fault detection capabilities to enhance overall fault tolerance. Abad et al. [16] studied the safety conditions for recovering software-faulty modules in cyber-physical systems, while Abdi et al. [17] proposed a system-wide restart method that leveraged system inertia to prevent destabilization. The efficacy of redundancy was studied and applied to individual neurons in trained NNs to increase fault tolerance [18], and Khunasaraphan et al. [19] developed a technique for quickly restoring weights and recovering the entire NN after fault detection.

In this work we plan to design a resilience scheme that recovers the full safety critical features of the ADV software stack through rejuvenation after having triggered upon fault detection an emergency trajectory execution mechanisms which steers the ADV into a safe spot.

The recovery method proposed in this paper belongs to the category of shallow recovery methods, which aim to repair faulty components of a CPS with minimal or no operation on the system states. For instance, Abad et al. developed a technique that restarts a failed component and replaces it with a healthy one [16], while Shin et al. suggest leveraging redundancy to fuse the output of multiple replicas and isolating and restarting the origin of the faulty contribution upon attack detection [20].

2 Emergency Recovery Scheme

2.1 System Model

We refer to a standard ADV architecture and which can be delineated as a succession of cascading modules:

- *Perception system* comprising sensors (cameras, LIDAR, localization) and computer vision tools to first fuse different sensor modalities and then detect, classify surrounding obstacles.
- *Prediction module* predict through the use of trained neural networks, the trajectories of previously detected obstacles.
- *Planning task* outputs a spatio-temporal ADV trajectory latest prediction output and a selected destination point.
- *Control task* computes the required steering, braking and throttling commands in order to execute the received ADV trajectory in a stable fashion.
- *Electronic control unit (ECU)* actuates the computed control commands.

2.2 Fault Model

In analyzing the ADV architecture shown in the top half of Figure 2, it becomes clear that each module within the system represents a single point of failure. This means that any fault occurring in one module can result in either an incorrect computation by subsequent modules or a delay and/or absence in the transmission of information, ultimately leading to the disruption of the ECU's ability to generate accurate and timely control commands. GPU greedy algorithms such as obstacle classification and prediction that power the perception and prediction modules, make this part of the software stack not only safety-critical to the safe maneuver execution of the ADV but are also highly vulnerable to potential faults. We consider in this work only software based crash fault i.e. non-responsiveness in the perception and/or prediction modules or time delayed attacks [21] performed on any of the task which all together can lead to missing or delayed information forwarded to the subsequent modules. We assume that this type of fault can be detected with high coverage by setting a hard deadline on the periodic execution time of the control task.

2.3 Emergency Maneuver

In order to tolerate the type of crash fault laid out in out fault model without the need to investigate its source, we propose to instantiate a parallel and more lightweight software stack following a simplex architecture [22]. The latter is devoid of the fault-prone perception-prediction module complex and is composed of a simpler planning module and a replica of the original control module. We refer to the lightweight planning module as the simplex-planning module whose task is to compute at every moment a potential trajectory to the nearest safe spot e.g. an emergency lane on a highway or a parking spot (see Figure 1).

A switch reads at every processing cycle the output of both the original and simplex stack, and by default forwards the commands of the original module to the ECU. If a crash of the original stack is detected via a missing or delayed control

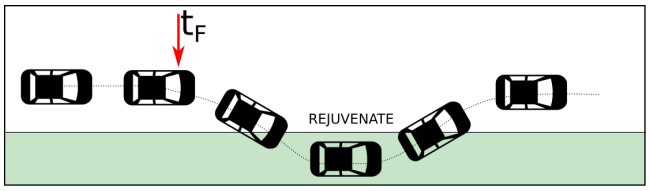

Figure 1: Emergency parking. A module crash is detected at time t_F which triggers the simplex-planning module to compute a trajectory into the safe spot area (green) where the whole autopilot software stack can be rejuvenated.

command, the switch forwards the control commands of the simplex stack to the ECU which steers the ADV into a safe spot. Once the car is at a given safe spot, a reboot signal is sent to the original software stack and consequently triggering a rejuvenation of the latter. This is schematically represented on Figure 2. While freshly rebooting a new stack during which all the software components have been re-instantiated from a trusted source (e.g. original NN weights are loaded into GPU memory and control algorithms are rejuvenated with new internal states) we exclude intermittent software faults to be resolved by this procedure.

To ensure the safe execution of this type of emergency maneuver, it is necessary to assume that the ADV is always near a safe lane, such as on a highway, and therefore cannot be performed in highly congested areas like city centers. With this assumption, it becomes feasible to compute an emergency trajectory at each processing cycle. However, in order to guarantee the dependability of switching from the original to the simplex stack devoid of perception system, the most recent computed trajectory has a short expiration time which should be significantly longer than the crash-fault detection time of the original stack. The specific duration of this expiration time depends on the speed of the ADV and the distance between the current and safe spot position. This will be investigated in the evaluation section of this manuscript.

We make the standard assumption that both stack, original and simplex are isolated and diverse enough, for example through obfuscation, to ensure that they fail independently with high coverage. Implementation-wise, this will be achieved via containerization (see next section). Our design employs a hybrid architecture, in which we differentiate the fault model of our trusted components in the simplex stack. While the components of the original stack are susceptible to crash failures, simplex-planning and the replicated control module must not fail. We justify this requirement based on their simplicity, as both components have considerably less number of lines of code than those in the original stack. Specifically, we assume that techniques such as ECC and scrubbing are in place to correct the effects of accidental faults in the stored data.

3 Implementation and Evaluation

The scheme proposed in the system model above will be implemented in the Apollo ADV software stack and simulated using the SVL physics simulator.

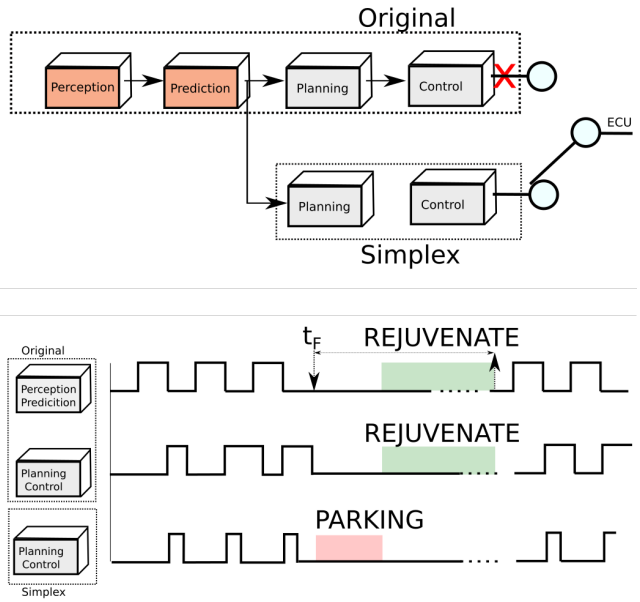

Figure 2: Top: Switching between the original and the simplex autopilot software stack once a crash (red boxes) in the original stack cause a delayed response from the original control module to the ECU. Bottom: Temporal logic of the recovery scheme presented above in which a fault is detected at time t_F and the parking scenario by the simplex stack is executed by the ECU. Green area denotes the time where the original stack is rejuvenated after which the normal driving conditions are restored.

Apollo utilizes containers to provide isolation and protection for its components. Containers provide a restricted execution environment with communication capabilities between containers and are hosted on top of a Linux-based operating system within Apollo. In a deployed system, we assume that critical component containers will be directly hosted on top of a real-time operating system (RTOS) capable of providing the necessary isolation. However, the RTOS in these architectures represents a single point of failure that will need to be addressed in the future (as demonstrated in the Midir architecture [23]). For demonstration purposes, we will implement both the original and simplex architecture of the software stack in two isolated containers, in which the inter-process communication will be assured via a simple TCP socket. Moreover, we plan to run the switching mechanisms between the two stacks in a third container for further separation.

In order to demonstrate our approach, we plan to design a set of diverse driving scenarios in SVL comprising a simple cruise control along a highway and a more complex situation inside a congested city area with multiple pedestrians and other vehicles. We will evaluate the efficacy qualitatively i.e. the ability of the ADV to avoid sudden crashes with surrounding obstacles. We will simulate a crash fault of the original stack by stopping the processing of any of the original stack submodules.

As we have laid out in out fault model section, the dependability of the emergency maneuver can only be guaranteed if

the crash fault detection delay is significantly shorter than the expiry time of the planned emergency trajectory. By simulating a whole range of scenarios, we will study the dependency of the crash-fault detection delay and the ADV speed at the moment of the crash on the efficacy of the emergency maneuver.

4 Acknowledgments

This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).

References

[1] "Toyota "unintended acceleration" has killed 89," May 2010.

[2] A. Lima, F. Rocha, M. Völp, and P. Esteves-Veríssimo, "Towards safe and secure autonomous and cooperative vehicle ecosystems," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pp. 59–70, 2016.

[3] Z. Peng, J. Yang, T.-H. P. Chen, and L. Ma, "A first look at the integration of machine learning models in complex autonomous driving systems," 2020.

[4] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[5] G. F. Elsayed, I. Goodfellow, and J. Sohl-Dickstein, "Adversarial reprogramming of neural networks," *arXiv preprint arXiv:1806.11146*, 2018.

[6] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1625–1634, 2018.

[7] H. Aghakhani, D. Meng, Y.-X. Wang, C. Kruegel, and G. Vigna, "Bullseye polytope: A scalable clean-label poisoning attack with improved transferability," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 159–178, IEEE, 2021.

[8] B. S. Arad and A. El-Amawy, "On fault tolerant training of feedforward neural networks," *Neural Networks*, vol. 10, no. 3, pp. 539–553, 1997.

[9] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, "Dirty road can attack: Security of deep learning based automated lane centering under {Physical-World} attack," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 3309–3326, 2021.

[10] M. Panoff, R. G. Dutta, Y. Hu, K. Yang, and Y. Jin, "On sensor security in the era of iot and cps," *SN Computer Science*, vol. 2, no. 1, pp. 1–14, 2021.

[11] C. Zhou, Q. Yan, Y. Shi, and L. Sun, "Doublestar: Long-range attack towards depth estimation based obstacle avoidance in autonomous systems," *arXiv preprint arXiv:2110.03154*, 2021.

[12] M. Darms, P. Rybski, and C. Urmson, "Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 1197–1202, IEEE, 2008.

[13] K. Geng and S. Liu, "Robust path tracking control for autonomous vehicle based on a novel fault tolerant adaptive model predictive control algorithm," *Applied Sciences*, vol. 10, no. 18, p. 6249, 2020.

[14] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, "SVL Simulator: A High Fidelity Simulator for Autonomous Driving," *arXiv e-prints*, p. arXiv:2005.03778, May 2020.

[15] I. Iso, "Pas 21448-road vehicles-safety of the intended functionality," *International Organization for Standardization*, 2019.

[16] F. A. T. Abad, R. Mancuso, S. Bak, O. Dantsker, and M. Caccamo, "Reset-based recovery for real-time cyber-physical systems with temporal safety constraints," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2016.

[17] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 10–21, IEEE, 2018.

[18] L.-C. Chu and B. W. Wah, "Fault tolerant neural networks with hybrid redundancy," in *1990 IJCNN international joint conference on neural networks*, pp. 639–649, IEEE, 1990.

[19] C. Khunasaraphan, T. Tanprasert, and C. Lursinsap, "Recovering faulty self-organizing neural networks: By weight shifting technique," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, vol. 3, pp. 1513–1518, IEEE, 1994.

[20] J. Shin, Y. Baek, J. Lee, and S. Lee, "Cyber-physical attack detection and recovery based on rnn in automotive brake systems," *Applied Sciences*, vol. 9, no. 1, p. 82, 2018.

[21] K. Xiahou, Y. Liu, and Q. Wu, "Robust load frequency control of power systems against random time-delay attacks," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 909–911, 2020.

[22] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 99–107, IEEE, 2009.

[23] I. P. Gouveia, M. Völp, and P. Esteves-Verissimo, "Behind the last line of defense: Surviving soc faults and intrusions," *Computers & Security*, vol. 123, p. 102920, 2022.