2023 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
MODELING SIMULATION & SOFTWARE (MS2) TECHNICAL SESSION
AUGUST 15-17, 2023 - NOVI, MICHIGAN

EXPLORING THE IMPACT OF DATA UNCERTAINTIES IN AUTONOMOUS GROUND VEHICLE PLATOONING

August St. Louis and Jon C. Calhoun

Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC

ABSTRACT

To improve robustness of autonomous vehicles, deployments have evolved from a single intelligent system to a combination of several within a platoon. Platooning vehicles move together as a unit, communicating with each other to navigate the changing environment safely. While the technology is robust, there is a large dependence on data collection and communication. Issues with sensors or communication systems can cause significant problems for the system. There are several uncertainties that impact a system's fidelity. Small errors in data accuracy can lead to system failure under certain circumstances. We define stale data as a perturbation within a system that causes it to repetitively rely on old data from external data sources (e.g. other cars in the platoon). This paper conducts a fault injection campaign to analyze the impact of stale data in a platooning model, where stale data occurs in the car's communication and/or perception system. The fault injection campaign accounts for different occurrences of a communication error. Our analysis provides an understanding of the sensitivity of each model parameter in causing system failures (e.g. a crash between vehicles within the platooning model). By understanding which parameters are most influential to the fidelity of the model, we enable the ability to make platooning algorithms safer.

Citation: A. St. Louis and J. C. Calhoun, "Exploring the Impact of Data Uncertainties in Autonomous Ground Vehicle Platooning," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 15-17, 2023.

1 INTRODUCTION

In recent years, intelligent transportation systems (ITS) have seen several advancements in collision avoidance and safety improvements [1]. The increased robustness of perception and communication technologies have allowed for a better situational awareness between vehicles [2]. Better situational awareness has allowed for the implementation of autonomous vehicle platooning. The vehicles in the platoon can not only understand their own surroundings, but are able to understand the states of the other vehicles in the platoon. While this provides each vehicle with a more holistic view of its environment, it also requires communication between each vehicle. While navigating the environment, vehicles communicate certain parameters such as speed, acceleration, and position to following vehicles. Following vehicles calculate their next step by considering other vehicles' current parameter measurements and adjusting its own to follow behind at a safe distance and speed. This update needs to occur every second, as changes in conditions can occur in an instant. This introduces a significant problem, what if something interrupts that flow of information? A connection issue can cause a vehicle to go blind to the other vehicles in the network. The longer the communication interference, the greater the chance the vehicle inevitably crashes. Connection issues arise in many ways, including: denial of service (DoS), false data injection, and modification attacks [3]. These systems depend greatly on communication between vehicles to account for obstacles in the environment. A perturbation of as little as a couple seconds can be enough to cause a crash.

1.1 Stale Data

There are several vulnerabilities that, if exploited, negatively influence the autonomous vehicle platooning system. Perception and communication devices are integral to a platooning system. If there is interference within one of those devices, the system has problems updating. Significant vulnerabilities include interfering with a vehicle's electric control unit (ECU). The ECU controls data processing and connection between the vehicle and other entities [3]. Disrupting the ECU causes errors in both perception and communication. Jamming attacks prevent sensor information from being translated to the ECU, false data injection attacks send spoofed information to the ECU, and a denial of service (DoS) attack bombards the ECU with too much information, making the ECU incapable of collecting data from vehicle sensors [3]. In response to an interference, the system commonly returns the last known values for perturbed data; this is called stale data. Stale data refers to a system with data that is not updating regularly. Systems with stale data commonly experience errors due to inaccurate assumptions based on incorrect data. Figure 1 displays the effects of stale data on a 1D platooning model. The effects of stale data on a 1D platooning model are displayed in the blue car. Originally, all the vehicles are moving at a constant 20 m/s, 20 meters apart. As the vehicles increase their velocities to 30 m/s, the blue car's velocity fails to update and remains unchanged. This causes the car to have uneven spacing and in the worst case will lead to an eventual crash.

1.2 Contributions

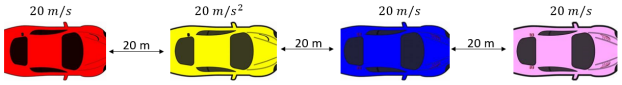
The goal of this paper is to analyze the effects of introducing stale data into a 1D platooning model. We inject perturbations into the model to test which model variables are most sensitive to a perturbation; that is, which variable, if perturbed, most likely results in a model failure. Observing aggregate simulation results allows us to discover which variables and variable settings are the most sensitive. Determining the sensitive part of the model enables development of measures to make the model more fault-tolerant. This paper makes the following contributions:

- Presents a portable methodology for injecting

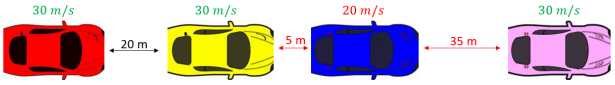
stale data into a MATLAB/Simulink model.

- Analyzes impact of stale data in a 1D platooning model.
- Results show our 1D platooning model is sensitive to stale data, with 33% of simulation instances ending in collision.
- Conveys that sensitivity varies based on which model variable is observed.

The remaining sections of this paper are as follows. Section 2 presents a background on vehicle platooning, stale data, and fault injection. Section 3 introduces our fault injection methodology and our evaluation metrics. Section 4 provides a detailed analysis of the impact of stale data on a 1D platooning model. Section 5 discusses study trends. Section 7 concludes the paper and provides future work considerations.



(a) Platoon before stale data.



(b) Platoon after stale data.

Figure 1: 1D platooning model before and after stale data.

2 BACKGROUND

2.1 Vehicle Platooning

Current implementations of intelligent transportation systems are becoming increasingly robust, but are not perfect. Several accidents relating to self-driving vehicles occur because of a lack of knowledge about that vehicle's surroundings [4]. In this work, researchers have explored several methods of mitigating the dangers associated with self-driving vehicles. One of which is cooperative driving. A common cooperative driving application is called Cooperative Adaptive Cruise Control (CACC). CACC and cooperative driving are common implementations of vehicle platooning. Applications of CACC use vehicle-to-vehicle communication (V2V). This methodology allows a vehicle to obtain information from a preceding vehicle in order to inform its own next step. This allows a system of vehicles to better anticipate problems and react quickly to those problems. Possible problems that may arise are influenced by several unknowns. Significant unknowns include adverse weather conditions, physical obstacles, and foreign entities to the network (e.g. other vehicles, animals, or humans). CACC has a significant positive effect on traffic safety and efficiency [5].

The vehicles within an Autonomous Vehicle Platoon (AVP) are split up into two groups, platoon leader (PL) and platoon followers (PF) [6]. Newer implementations of AVPs use an effective Reputation-based Leader Election scheme that observes the trustworthiness of each vehicle in the platoon based on past actions and trips. This framework decides which vehicle becomes the PL. The PL has the greatest responsibility and has a direct influence on the actions of the platoon. The PL is tasked with dynamically monitoring road conditions, collecting and processing information, and issuing driving instructions to PFs [6]. Utilizing a vehicle platoon increases fuel efficiency by greatly decreasing wind resistance to the PFs [6]. To offset the increased drag on the PL, the PFs share some of the fuel they have saved from reduced wind resistance. While several aspects of AVPs are not directly implemented in our 1D platooning model, the topics discussed in this section greatly influence real life vehicle platooning implementations.

2.2 Stale Data

Self-driving operating mechanisms are controlled and monitored by computer-based algorithms [7]. Data and information fidelity are nontrivial aspects of intelligent transportation systems. Attacks on intelligent systems require insight into the failure conditions of the equipment, control principles, process behavior, signal processing, etc. [8]. In this work, Krotofil et al. discuss how attackers introduce stale data into systems. They describe how that interference propagates in other areas of the system. Programmable logic controllers (PLC) are entities that are used as automation controllers. PLCs operate in a scan cycle architecture, meaning their control logic uses the last saved input values to relay commands to actuators [8]. An attacker interferes with a PLC by jamming its sensor input readings, forcing the system to continuously read in the same values. Attackers also jam the connection between the PLC and an actuator, allowing the state of the controller to update but blocking the system from actually acting on the update [8]. Several of these principles are common to stale data attacks. The way in which we inject stale data into our model is largely similar to the methodologies described in Krotofil et al. [8].

2.3 Fault Injection

Faults introduce errors into a system and are categorized as either hard or soft [9]. A hard fault is systemically reproducible. An example of a hard fault is the inability to communicate to a vehicle that is offline. A soft fault is a fault where activation is not systematically reproducible. These errors are often transient, such as dropped messages and data corruption via cosmic radiation [10].

As integrated circuit designers and manufacturers explore more robust technologies in circuit design, sensitivity in these circuits becomes a non-trivial issue [11]. In this work, the authors describe the necessity of dependability analysis in combating several natural and deliberate perturbations. These perturbations are examples of system faults; faults manifest in a number of different ways. Particle strikes and electromagnetic interference are examples of natural system perturbations. The presence of natural phenomena can result in faulty logical behavior and possibly application failures. A deliberate fault-based attack can include lasers that are utilized to hack critical data stored in circuits, such as cryptographic keys and other security features. The presence of a fault can result in application failure either from an erroneous value induced on a circuit output, or from an erroneous sequential behavior due to one or more incorrect bits in internal registers [11]. These internal errors are defined as soft errors.

In order to inject faults into a computing system, software based fault injectors represent low-cost and flexible methods by corrupting values in the executing binary [12], values at the register level using a compiler [13], or perturbing communication [14]. In this work, we inject stale data into our simulation at a software level using a Simulink module.

3 METHODOLOGY

3.1 Model Introduction

Xuan and Naghnaeian [15] present a mathematical formulation of a 1D platooning model that includes a lead vehicle and a variable number of follower vehicles. The model updates continuously, relaying information from each vehicle throughout the system. Figure 2 displays the vehicle features for the follower and lead vehicles: relative distance, position, velocity, throttle input, and acceleration. All of these features are logged at runtime and stored for analysis subsequent analysis. These values are used in the governing mathematical equations to calculate the current states of the other vehicles in the model. The updated information is integral in allowing each vehicle to *see* the vehicle to its anterior, as the results of these equations control the simulation's trajectory.

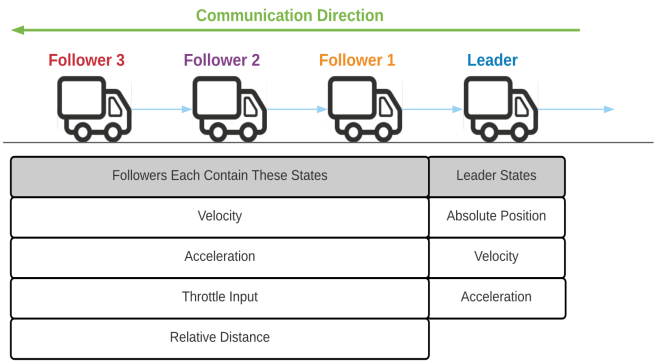


Figure 2: Vehicle features of 1D platooning model from [15].

3.2 Stale Data Injection

The model simulates vehicles traveling down a road in 1D with a fixed distance between each vehicle. Each vehicle communicates with the vehicle to its immediate anterior. As discussed before, it is possible that data will be communicated late or not at all. At any juncture, and for any duration, the model can experience a disruption. In the occurrence of a lapse in data, the model relays stale data. In order to inject stale data into the model, we implement a custom Simulink block written in MATLAB. This Simulink block is placed on the input signals of our platooning model [16]. This allows us to simulate stale data injections by essentially turning off the model's input for a specified duration. This method is portable within the MATLAB and Simulink environments and can be applied to other simulations. Observing the perturbed model output allows us to quantify the effect of specific stale data injections that occur at any time-step and for any duration.

Understanding the error propagation associated with different stale data injected at various locations and times allows us to understand not only sensitive model variables, but also important injection junctures and durations. Error propagation impacts the future states of the vehicles; therefore, it is critical we determine when error propagates through the system and when it attenuates. For example, if the acceleration of follower vehicle two is perturbed, and stale data is introduced, the model calculates incorrect values for connected model features. The model sustains the same acceleration value instead of the model calculated next step acceleration value. The relative distance, velocity, and throttle input displays values complimentary to the incorrect, repeated acceleration value. Errors are likely to propagate within the other features of the same vehicle. In most cases, the error is likely to be passed on to the following vehicles as they adjust themselves to perturbed feature values.

3.3 Evaluation Metrics

In order to properly analyze the output of a simulation instance, we must define what constitutes a simulation's success or failure. To classify a simulation as a success or failure, we determine whether there has been a crash between any vehicle. A successful simulation features no crashes, while a failed simulation features one or more crashes. In order to determine the presence of a crash, we identify relative distance between vehicles as our evaluation metric. Relative distance is a variable unique to each follower vehicle that represents the distance in meters a vehicle trails the vehicle to its immediate anterior. If at any point in a simulation, the relative distance of a vehicle falls is zero or less, that vehicle has crashed into the vehicle in front of it.

The goal of injecting stale data into the model is to determine what variables are most sensitive. This means which variables, if perturbed, result in the most model failure. To determine which model variables are most sensitive, we observe the frequency of crashes for each variable. This allows us to quantify which variables have a significant effect on the success or failure of the simulation. Relative distance is considered over other metrics because it is an all-encompassing metric for our purposes. Variations in other metric results in a zero relative distance if the changes are significant

enough. For example, if vehicle acceleration is perturbed drastically but does not result in a vehicle crash, the model eventually returns to a steady state.

As explained above, the most important evaluation metric we consider is relative distance. As we discover sensitive model variables, we are essentially defining which model variables have the most effect on relative distance. Another method of quantifying a variable's effect on relative distance is to observe the magnitude of error introduced into the relative distance after a perturbation. We calculate the difference in relative distance values in a perturbed simulation to the baseline fault-free simulation.

3.4 Defining Perturbations

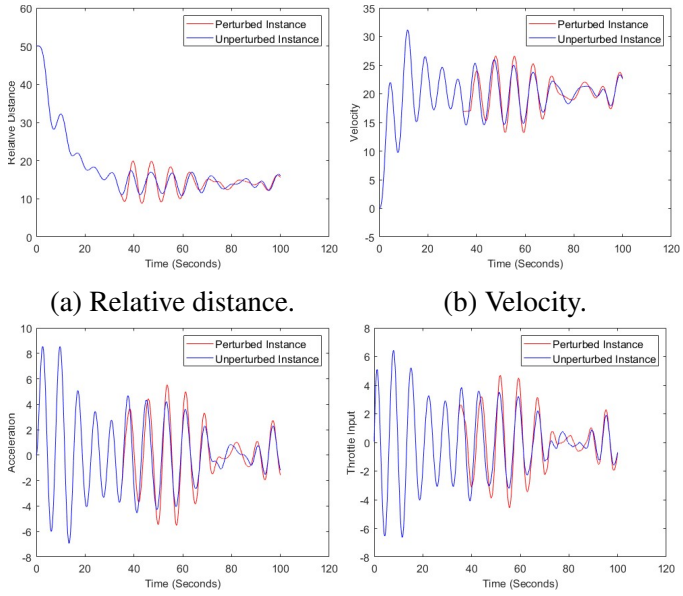
We perturb simulations of the model to understand which model parameters are the most sensitive. In order to simulate perturbations, we inject stale data into the model. There are three perturbation parameters we use to determine the location and duration of the perturbation: Perturbation Juncture (PJ), Perturbation Duration (PD), and Perturbation Variable (PV).

- **Perturbation Juncture (PJ):** represents a time-step where the perturbation begins.
- **Perturbation Duration (PD):** determines how long the perturbation lasts.
- **Perturbation Variable (PV):** signifies the model feature that is perturbed.

At a random time (PJ) in the model, the values for a feature (PV) will not update for an arbitrary duration (PD). Throughout that duration, the value of that variable remains constant until the injection is complete.

Figure 2 states the significant vehicle features that are recorded and updated continuously over the course of the simulation. The lead vehicle records its absolute position, velocity, and acceleration. The following vehicles record their velocity, acceleration, throttle input, and relative distance to its preceding vehicle. This data is stored in a table that depicts the changes in each feature for each time-step throughout the simulation. We use the data in this table to determine the success or failure of each simulation. Moreover, we use it to determine precisely what happened and/or went wrong in the simulations.

Each perturbation propagates error differently throughout its own vehicle and to others. Figure 3 displays an example of how a perturbation introduces stale data into the model (i.e. visible in near 40 seconds in the velocity graphic where the velocity stays constant for a short amount of time). The presence of stale data skews model values within multiple model features as the error propagates throughout the model. However, we see the noise dampens as the model seeks to obtain its steady state solution with the fixed following distance [16].



(a) Relative distance. (b) Velocity.

(c) Acceleration. (d) Throttle input.

Figure 3: The propagation of an error in the vehicle features of follower vehicle 1 due to a perturbation.

4 EXPERIMENTAL RESULTS

4.1 Model Details

The 1D platooning model is run in a MATLAB and Simulink environment. The model is a simulation of a group of four vehicles driving autonomously in a straight line on a road. The model simulates the vehicles through arbitrary start and end points, where the vehicles accelerate to a designated speed, 20 meters per second, while sustaining 20 meters of space between each other. The four vehicles are categorized into two groups, a leader and its followers. A follower vehicle receives the variable settings from the car directly in front of it. Then, considering those values and its own, updates its variables in the next time-step in order to sustain relative distance requirements.

In order to test for multiple perturbation scenarios, we run 10,000 instances of the model. The three perturbation parameters, selected at random, define the unique fault injection. The PJ is a random number from 1 to 102; this number represents the time-step during the simulation that the perturbation begins. The PD is a duration between 1 and 10 seconds, this is the number of time-steps where stale data is entered into the model. The PV is a random number between 1 and 15, each number representing a different model feature to be perturbed. We run all experiments on a Windows 11 workstation with an Intel i9-12900K processor with 64.0 GB of RAM. The platooning model runs in MATLAB version 9.12.0 and Simulink version R2022a Update 1.

4.2 Model Results

To understand how sensitive the 1D platooning model is to stale data, we first examine high-level observations from the fault injection campaign. At a high level, we look for results that highlight holistic model trends. After each run completes, we classify it as success or not based on if any collisions occur. Results show that 33.8% of runs result in a failure. This information is pertinent to understanding model trends, but lacks specificity. To understand what causes model failures, we look to attribute perturbation parameters to failures. To understand which parameter settings influence failures the most, we observe four metrics: (1) Percentage of failures per instance, grouped by vehicle feature; (2) Average number of crashes per perturbation, grouped by vehicle feature; (3) Average model time to failure, grouped by vehicle feature; and (4) Error propagation patterns for the vehicle's position in response to a perturbation.

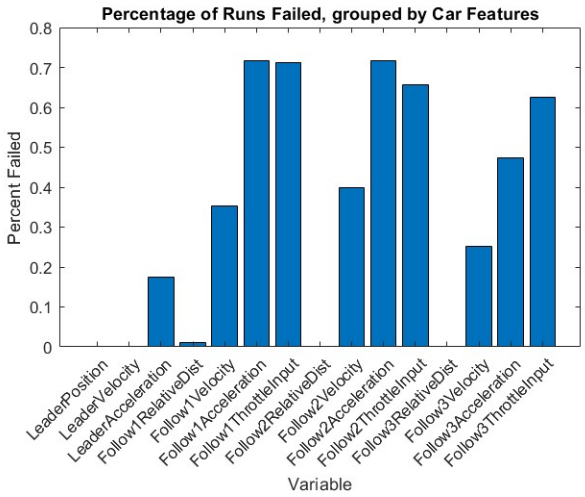


Figure 4: Percentage of failures by model feature.

4.2.1 Percentage of Failures

By summarizing our output data, we are able to depict the count of failures per each perturbation juncture. Figure 4 shows the percentage of runs where at least one vehicle crashes when a specific feature is perturbed. This information is important because it allows us to pinpoint the most significant model features that lead to failures. The common trend among the followers is that acceleration and throttle input are the most sensitive features, as stale data in them leads to crashes nearly 50% of the time, regardless of the vehicle perturbed. Follower 3 has very high fail rates due to its placement in the platoon, resulting in possible corruption when any vehicle suffers stale data. Knowing

each model feature's failure percentage provides important information about the sensitivity of each feature. Understanding more model trends provides another level of analysis to base conclusions off. Each subsequent section includes results that make it easier to classify feature sensitivity.

4.2.2 Average Number of Crashes

Not all failures are created the same. By adding additional specificity to our definition of sensitivity, it is possible to further differentiate model features with similar model sensitivities. A failure is characterized as at least one vehicle crashing into another. However, there are several cases where more than one vehicle crashes. Although these cases result in the same outcome, a model failure, it is important to differentiate between the two situations. Furthermore, it is possible that two variables have similar fail rates for single vehicle crashes, but different fail rates for multiple vehicle crash scenarios. A feature that causes more total crashes is a more sensitive model feature.

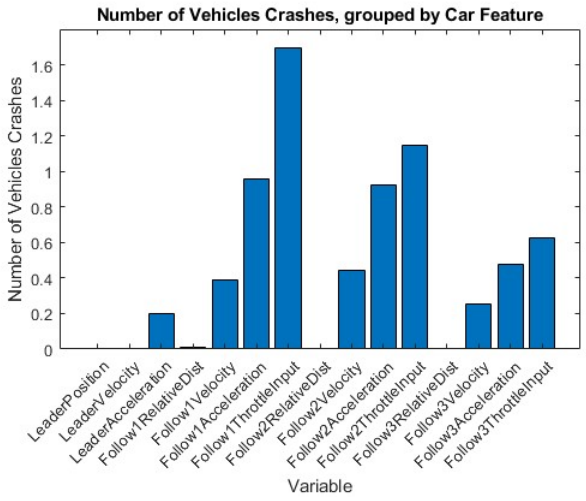


Figure 5: Number of crashes by car feature.

Figure 5 displays the average number of crashes per perturbed variable. Follower 1's throttle input is the most sensitive feature here by far, with 1.6 crashes on average. Other features are much less sensitive. Moreover, stale data at the front of the platoon is the most sensitive as it can propagate to all vehicles, leading to the higher crash rate. Vehicles at the end of the platoon see the lowest crash rate on average.

4.2.3 Time to Failure

Time to failure is an important metric because it gives further insight into the sensitivity of model features. A failure that occurs after a long amount of time may not occur if injected late enough into a simulation. Additionally, that failure could possibly be avoided if the perturbation duration was decreased. Therefore, a feature with a short time to failure is sensitive to the model. This means that a feature with a short time to failure effects the model significantly in a short time frame. Furthermore, for certain features, the error propagates throughout the model quickly. This usually means the error is propagating quickly within the original vehicle. It can also mean that error is propagating quickly vehicle to vehicle, causing errors in multiple vehicles, further shortening the time for a crash to occur.

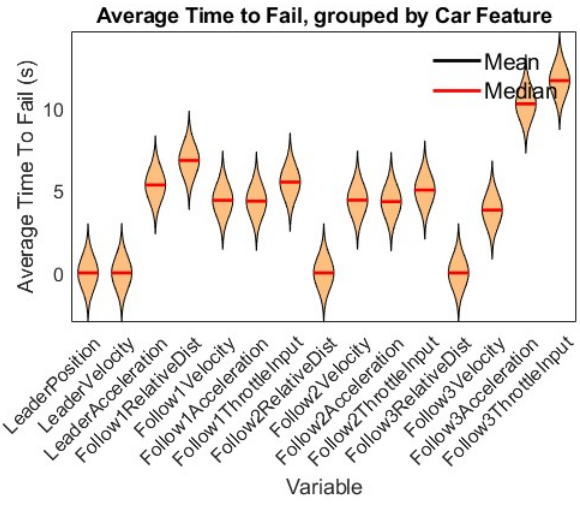


Figure 6: Average time to failure for car feature.

Figure 6 displays the average time to failure values for each variable. Velocity, acceleration, and throttle input are comparable in most cases, except for Follower 3. We believe the large difference in Follower 3 is due to it only neighboring one other vehicle. Without error propagation to other vehicles, the time to failure is extended. Excluding those outliers, observing this graphic does not give a clear, conclusive answer to the most sensitive model feature in relation to average time to failure.

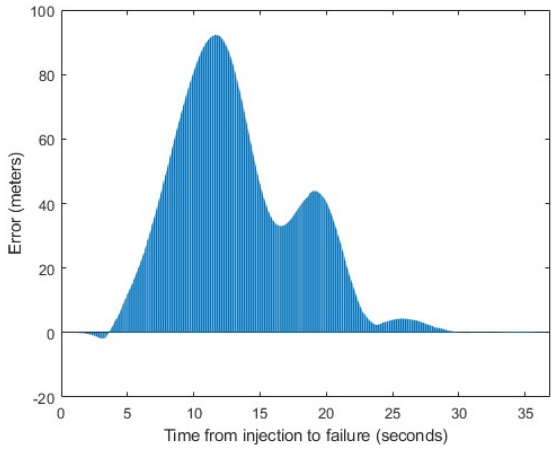
4.2.4 Error Propagation

In order to properly depict the sensitivity of each variable, an understanding of the error propagation is an integral means of comparing variable sensitivity. The first three result metrics allow us to observe the error propagation within the model. By observing Figure 3, we see that a perturbation below a certain threshold negatively affects the system, but eventually, the system self-corrects and reverts to normal. We define this threshold as the minimum amount of error needed for model failure. Therefore, Figure 4, the model features that propagate enough error to cause a failure. Figure 5 displays the magnitude of error introduced into the model for each feature. A larger number of crashed vehicles means that there is a large amount of error being propagated throughout the system, enough to make multiple vehicles crash. Figure 6 shows how fast errors propagate through the system for each model feature.

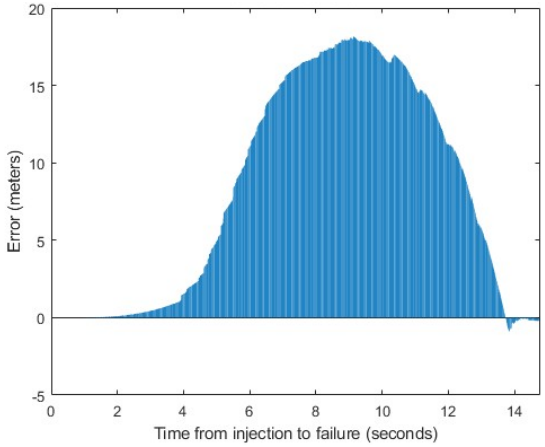
All model results are in some way related to the patterns associated with error propagation; therefore, the information provided by error propagation plots is significant. By observing error propagation, we compare exactly how each variable negatively affects the model's output. Figure 7 displays the propagation of error through multiple vehicles when the Follower 1 has its throttle input perturbed. This is the average error in relative distance for the time period after a perturbation occurs. We see two patterns occur. The pattern in Follower Vehicle 1 is how the error usually propagates in the vehicle where the perturbation occurs. The error spikes, then slowly returns to zero after a couple peaks. The algorithm notices error within the data and tries to adjust the relative distance readings to their correct values. The peaks likely represent the algorithm's attempts to fix the relative distance, while the values change in the opposite direction. For example, the algorithm knows that it needs to change data to depict true value, so it may be decreasing relative distance values when the algorithm would normally increase relative distance at that time interval. This would cause a small spike in error. The other followers have a single peak of error that returns to zero after some time. The absence of multiple peaks is likely because the algorithm does not try to self-correct errors passed onto other vehicles, it allows those vehicles to fix themselves through additional calculations. The most important aspect of the error propagation graphics is the magnitude of the initial error peak. The higher the error, the greater the chance of model failure. Whichever feature generates the most error should be the most sensitive model feature. After comparing the magnitudes of all model features, we have observed that throttle input in Follower Vehicle 1 has the greatest magnitude of error on average. Therefore, this feature is the most sensitive.

5 DISCUSSION

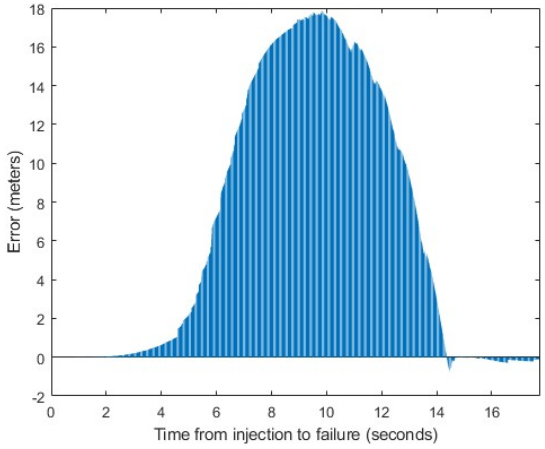
Observing the model results, we make a couple of conclusions about the most sensitive model variables. Based on our results, we conclude that the most sensitive model features is vehicle throttle input. Regardless of vehicle, a perturbation in one of these model features consistently returns the highest rates of failures in the model and the highest number of car failures per iteration. Additionally, observing the error propagation graphics for these features convey a consistently higher magnitude of error for their perturbations.



(a) Follower Vehicle 1.



(b) Follower Vehicle 2.



(c) Follower Vehicle 3.

Figure 7: An example of the error propagation through different vehicles.

6 RELATED WORKS

Lin et al. [17] provides a comprehensive overview of the world of ITS. They introduce several problems apparent in transportation now and how ITS alleviates them. The work gives a synopsis on the architecture of ITS, key technologies, and current challenges and opportunities in the area. Deng [18] observes the effects of heavy-duty vehicles (HDVs) on traffic flow. His work focuses more on the interaction of platooning vehicles to the surrounding environment. The framework used is complex and utilizes ACC/CACC algorithms. Applying stale data injections to a complex model like this would introduce additional factors to quantify model failure. Now the vehicles would have to consider other vehicles on the road, making a stale data attack more significant. Jin et al. [19] observes the macroscopic interactions between vehicle platoons and background traffic at highway bottlenecks. This work features multiple platoons operating at once. It is interesting to consider the implications of stale data on multiple communicating vehicle platoons.

7 CONCLUSION AND FUTURE WORK

This paper highlights the importance of robust ITS platooning systems that have sufficient situational awareness and fault tolerance. We introduce a 1D platooning model, in which we introduce perturbations into to simulate stale data. The model is simulated tens of thousands of times, introducing varying perturbation variables with each run. Our analysis shows that the most sensitive model features are vehicle acceleration and throttle input.

Future work includes extending our work to a 2D platooning model. The addition of a multidimensional model provides a more complex model equation. This will potentially change the importance of some model features, and subsequently their sensitivity to perturbations. In addition, we will explore techniques to detect and recover from stale data.

ACKNOWLEDGMENTS

This work was supported by Clemson University's Virtual Prototyping of Autonomy Enabled Ground Systems (VIPR-GS), under Cooperative Agreement W56HZV-21-2-0001 with the US Army DEVCOM Ground Vehicle Systems Center (GVSC).

References

[1] V. Milanés, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, and M. Nakamura, "Cooperative adaptive cruise control in real traffic situations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 1, pp. 296–305, 2014.

[2] R. Sengupta, S. Rezaei, S. E. Shladover, D. Cody, S. Dickey, and H. Krishnan, "Cooperative collision warning systems: Concept definition and experimental implementation," *Journal of Intelligent Transportation Systems*, vol. 11, no. 3, pp. 143–155, 2007. [Online]. Available: https://doi.org/10.1080/15472450701410452

[3] Y. Fraiji, L. Ben Azzouz, W. Trojet, and L. A. Saidane, "Cyber security issues of internet of electric vehicles," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.

[4] A. Farag, A. Hussein, O. M. Shehata, F. García, H. H. Tadjine, and E. Matthes, "Dynamics platooning model and protocols for self-driving vehicles," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 1974–1980.

[5] B. van Arem, C. J. G. van Driel, and R. Visser, "The impact of cooperative adaptive cruise control on traffic-flow characteristics," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 4, pp. 429–436, 2006.

[6] Z. Ying, M. Ma, Z. Zhao, X. Liu, and J. Ma, "A reputation-based leader election scheme for opportunistic autonomous vehicle platoon," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 3519–3532, 2022.

[7] I. J. Rudas and J. Fodor, "Intelligent systems," *International Journal of Computers, Communications & Control*, vol. 3, no. 3, pp. 132–138, 2008.

[8] M. Krotofil, A. Cárdenas, J. Larsen, and D. Gollmann, "Vulnerabilities of cyber-physical systems to stale data—determining the optimal time to launch attacks," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 4, pp. 213–232, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874548214000638

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[10] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *Electron Devices, IEEE Transactions on*, vol. 26, no. 1, pp. 2–9, Jan. 1979. [Online]. Available: http://dx.doi.org/10.1109/T-ED.1979.19370

[11] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 502–506.

[12] D. Li, J. S. Vetter, and W. Yu, "Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool," in *Proceedings*

of the International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 57:1–57:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389074

[13] J. Calhoun, L. Olson, and M. Snir, "Flipit: An llvm based fault injector for hpc," in *European Conference on Parallel Processing*. Springer, 2014, pp. 547–558.

[14] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 78:1–78:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389102

[15] Y. Xuan and M. Naghnaeian, "Detection and identification of cps attacks with application in vehicle platooning: a generalized luenberger approach," in *2021 American Control Conference (ACC)*, 2021, pp. 4013–4020.

[16] C. Holt and J. C. Calhoun, "Stale data analysis in intelligent transportation platooning models," in *2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2022, pp. 0313–0320.

[17] Y. Lin, P. Wang, and M. Ma, "Intelligent transportation system(its): Concept, challenge and opportunity," in *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)*, 2017, pp. 167–172.

[18] Q. Deng, "A general simulation framework for modeling and analysis of heavy-duty vehicle platooning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3252–3262, 2016.

[19] L. Jin, M. Čičić, S. Amin, and K. H. Johansson, "Modeling the impact of vehicle platooning on highway congestion: A fluid queuing approach," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*, ser. HSCC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 237–246. [Online]. Available: https://doi.org/10.1145/3178126.3178146