An Empirical Study of Reliability Analysis for Platooning System-of-Systems

Sangwon Hyun, Lingjun Liu, Hansu Kim, Esther Cho, Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
{swhyun, riensha, hansukim, esthercho, bae}@se.kaist.ac.kr

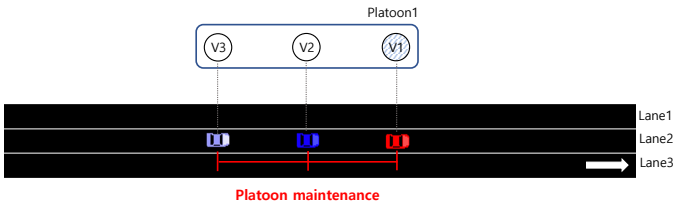
***Abstract*—As a type of advanced autonomous vehicle software, platooning system-of-systems (SoS) has received tremendous attention as a next-generation system. Platooning SoS can attain several benefits, such as increasing fuel efficiency and alleviating traffic congestion by grouping autonomous vehicles in close proximity. Many studies have focused on analyzing the reliability of platooning SoS, as it is a safety-critical system. However, existing studies have two major limitations in their reliability analysis: (1) the studies did not fully cover the internal uncertainties of platooning SoS, such as heterogeneity; (2) they restricted external uncertainties by limiting the test scenarios to a single platoon, which could adversely affect the confidence of the analysis results. In addition, there exists no common fault dataset for the analysis of platooning SoS. Therefore, we provide an open dataset for platooning SoS by considering internal and external uncertainty factors during simulations. We empirically analyzed the execution logs of random platooning SoS scenarios in terms of reliability. We found 16 types of failure scenarios and root causes of the failures, as a result of the empirical study. Further, we generated the benchmark dataset, PLTBench, by classifying all failed logs based on the detected failure cases. We provide all the artifacts and descriptions in our benchmark web page as well as example codes to utilize the PLTBench. The conclusions of this study can enrich the general failure scenarios and experimental data set of platooning SoS for future studies.**

***Index Terms*—Autonomous vehicle system, Platooning System-of-Systems, Cyber-Physical system, Empirical study, Simulation-based Reliability analysis, Benchmark dataset**

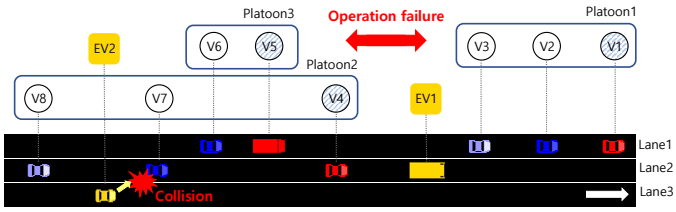
I. Introduction

Autonomous vehicle systems are indispensable when drawing next-generation outlook, and they are expected to play a major role in the next jump in transportation and accessibility. According to this trend, numerous studies have focused on developing advanced autonomous vehicle systems [1]–[5] and verifying quality issues (e.g., reliability and safety) of autonomous vehicles [6]–[10]. As a type of advanced autonomous vehicle system, platooning system-of-systems (SoS) has recently garnered considerable attention owing to various environmental and business benefits, such as increasing fuel efficiency and alleviating traffic congestion [11]–[14].

Platooning SoS which groups autonomous vehicles, as one unit in close proximity, has a positive impact on the fuel efficiency by aerodynamic benefits [15] and on the traffic congestion and travel times by increasing the road capacity [12]. The concept of SoS is conventionally applied in platooning systems [16]–[19] because a platoon consists of heterogeneous vehicles that have operational and managerial independence, and the vehicles autonomously choose whether to join or leave the platoon. The platooning SoS achieves the benefits by the interactions of constituent vehicles based on the platooning operations (e.g., `Join`, `Leave`, `Merge`, etc).



a Example simulation of existing analysis studies on platooning SoS [20]



b Example simulation of platooning SoS in this study

Fig. 1: Example simulations of platooning SoS in existing studies and in this study

Even with these benefits, because the platooning SoS is a kind of safety-critical system, it may have a huge adverse effect on various users when the SoS fails. Therefore, many studies have focused on analyzing the reliability of the platooning systems. However, we found that the majority of the existing studies focused on the reliability analysis of platooning SoS: (1) did not fully consider the major characteristics of the platooning SoS, such as heterogeneity and operational/managerial independence of constituent vehicles; (2) limited the testing and verification scenarios to a single platoon scenario simulation. The limitations badly affect the confidence of reliability analysis results on platooning SoS.

For instance, Elgharbawy [21], Kamali [20], and Meinke et al. [22] verified the platooning management protocol by simulating a single platoon consisting of homogeneous vehicles. Fig. 1a depicts an example simulation scenario based

on existing studies. A single platoon was generated in the simulation, and the maintenance of the generated platoon was checked. Despite that Kamali et al. [20] considered the environmental human-driven vehicles (HDVs), which add the external uncertainties in simulation, the existing studies did not fully cover the internal and external uncertainties of platooning SoS. Internal uncertainties (e.g., multiple platoon interactions, heterogeneous vehicles, and different autonomous driving policies) and external uncertainties (e.g., the existence of HDVs that cannot communicate with platoon vehicles) need to be considered in the analysis to increase the confidence in the analysis results. Shin et al. [18] released a physical testbed for platooning SoS and provided various goal properties. Nevertheless, the testbed does not consider the heterogeneity of vehicles and is limited to a single platoon scenario.

Additionally, there exists no common dataset of failures and faults in platooning SoS. The existing studies generated the experimental data by themselves based on different platooning simulators. The absence of such a common experimental dataset limits the quantitative performance evaluation of various research results in a unified experimental environment. Therefore, an experimental dataset to be used for the reliability analysis and verification of platooning SoS is needed.

To overcome the limitations, we propose empirical study results for the reliability analysis of platooning SoS and release an open benchmark dataset, PLTBench, including detailed failure scenarios and a completely labeled set of logs. The simulation logs analyzed in this study were generated by the multi-platoon and heterogeneous autonomous driving policy settings with environmental HDVs, as depicted in Fig. 1b. The PLTBench contains the raw logs of the simulations and categorization results of all failed logs by the failure scenarios, with a detailed description of the root causes. The major contributions of our study are as follows:

- We propose detailed results of the reliability analysis for the open platooning simulator, including 16 types of failure scenarios and their root causes.
- We provide an open benchmark dataset, PLTBench, consisting of raw logs and categorization results of all failed logs by the failure scenarios.

We generated 6,525 random platooning scenarios using the StarPlateS framework, which provides useful modules to utilize the open platooning simulator, VENTOS. In the analysis, we verified the execution traces using two verification properties related to reliability, *operation_success_rate* (*OSR*), and *collision_existence* (*COLL*). We found 10 types of failure scenarios that violate the *OSR* property and 6 code-level faults in VENTOS causing the failures, and 6 types of failure scenarios with 7 root causes at VENTOS codes in the *COLL* results. We manually classified all failed logs by the analyzed 16 failure classes and organized them as a benchmark dataset. We also increased the accessibility of the dataset by providing two example codes that utilize the dataset and their results. We expect that the analysis results and PLTBench could be utilized as generic failure scenarios and an experimental dataset in future studies targeting the platooning SoS.

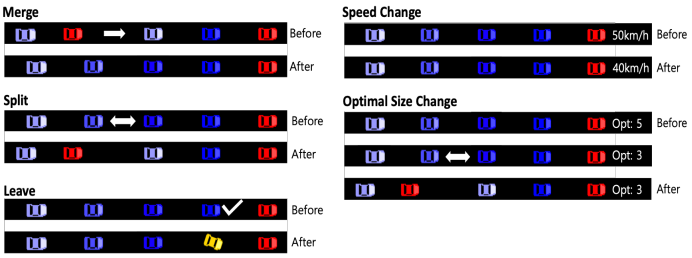


Fig. 2: Example platooning operations in VENTOS

The remainder of this paper is organized as follows. Section 2 explains the background. Section 3 describes the work related to this study. Section 4 elucidates the results of the empirical study on platooning SoS, and Section 5 presents the procedures and composition of the benchmark dataset. Section 6 explains the example utilization of the benchmark dataset. Finally, Section 7 discusses the implications of our findings and concludes the paper.

II. Background

A. VENTOS Platooning Simulator

VENTOS [23] is an open integrated transportation simulator that consists of a network simulator, OMNET++ [24], and a traffic simulator, SUMO [25]. It not only provides general vehicle simulation modules but also contains a platooning management protocol to simulate platooning SoS. The platooning management protocol in VENTOS provides five platooning operations: `Merge`, `Split`, `Leave`, `Speed Change`, and `Optimal Size Change`. Fig. 2 shows the illustrative examples of the five platooning operations. By the `Merge` operation, two platoons literally merge into one larger platoon. Once the merge request is approved, the red leader vehicle of the rear platoon becomes a follower in the front platoon. The `Split` operation splits one platoon into two smaller platoons when the leader of the platoon approves the request from the follower. The `Leave` operation is activated when a vehicle leaves the platoon after the platoon leader approves its request. Finally, the `Optimal Size Change` is performed when the platoon decides to change its platoon size, and the `Optimal Size Change` operation can cause a `Merge` or `Split` operation.

In this study, the open platooning management protocol provided by VENTOS is a target system for the reliability analysis. We utilized the StarPlateS framework to generate scalable experimental data using the platooning protocol.

B. StarPlateS Framework

StarPlateS [17] is a statistical verification framework for platooning SoS based on VENTOS simulator. It consists of three modules: scenario generation, execution, and verification modules. The scenario generation module generates random configurations and scenarios, which are inputs for VENTOS simulations, minimizing the number of scenarios that cause

run-time failures. Next, the execution module runs the configurations and scenarios on VENTOS simulator involving environment human-driven vehicles (HDVs) in simulation. The verification module checks the achievement rate of the given goals by using a statistical model checking technique.

To efficiently generate and verify the simulation traces of VENTOS platooning SoS protocol, we utilized the scenario generation and execution modules in StarPlateS. We also used the existing *OSR* property in the verification module and added a new *COLL* property to verify the logs.

III. Related studies

Numerous studies have been dedicated to the field of platooning systems, whereas few studies proposed testbeds for platooning SoS. Shin et al. [18] and Filho et al. [26] recently presented physical platooning testbeds to support the need for a physical experimental environment. Shin et al. [18] provided an open physical exemplar for platooning SoS, implementing platoon vehicles with LEGO robots. They evaluated several SoS goal properties (e.g., collision avoidance and road occupancy minimization) to demonstrate that vehicles adapted their driving behaviors to meet platooning SoS goals. The manuals of implementation, skeleton codes for LEGO vehicles, and the experimental results are fully accessible. Filho et al. [26] proposed a platooning robotic testbed with the integration of on-board units (OBU) and the robotic operation systems (ROS) on robotic vehicles. In their study, different cooperative control algorithms can be implemented in a container running on the ROS. In the experiments, they evaluated the following behavior based on the differences in the trajectories between a leader and followers. However, the accessibility of the developed testbed and its experimental results are limited. Although both studies considered the physical environmental uncertainties, the environmental vehicles were absent in their experiments. In addition, both testbeds are limited to a single platoon scenario that does not support multi-platoon scenarios and formed platoons with homogenous vehicles.

Several previous studies have been conducted on the reliability analysis of platooning SoS to verify specific properties of the platooning system through virtual simulation [20]–[22], [27]. Kamali et al. [20] proposed a verification technique for platooning systems by modularizing the system into an agent-based architecture. An automotive simulator, called TORCS, was utilized to implement the automotive environment; thus, environmental vehicles were considered in the simulation. They verified the spatial and real-time properties using the UPPAAL model checker [28] by abstracting the agent programs, including leader and follower vehicles, into timed automata. Vieira et al. [27] aimed at achieving platoon formations in more realistic scenarios and presented a platooning simulation framework that integrates an ROS-based simulator, Gazebo, with a network simulator, OMNET++. In this study, the communication between platoon vehicles was tested by providing different message sending frequencies to verify the maintenance of a platoon, such as the following behavior. Meinke [22] presented a case study of verifying the safety properties of platooning SoS to investigate the scalability of a learning-based testing technique. Meinke developed a JAVA-based simulator that generates single-platoon scenarios with different numbers of vehicles. This study focused solely on the scenarios, in which the leader vehicle changed its speed, thus the internal uncertainties of platooning SoS were partially considered. Elgharbawy [21] proposed a testing framework for an automated truck driving system to verify the safety properties in various test environments. They defined scenarios with different parameter settings for a vehicle and tested the driving behavior in critical scenarios. They considered the environmental perception errors in the system and defined the failure rate of environmental sensing with a stochastic property. However, this study does not empirically evaluate the performance of the framework. All these studies [20]–[22], [27] did not cover the heterogeneity of platoon vehicles, and the scale of their verification results was limited to the single platoon scenarios. Except for [20], these studies did not consider the external uncertainties, such as the environmental vehicles in simulation. Moreover, they generated the simulation scenarios by themselves to verify the system, and the experimental data were not fully released.

In contrast to the existing studies, our empirical study covers both the internal and external uncertainties. First, the study generated scenarios with various numbers of platoons. Diverse types of vehicles and autonomous driving models were utilized in this study. In addition, the stochastic environmental vehicles are included in the simulated scenarios.

IV. Empirical Analysis of Platooning SoS

In this section, we describe the empirical analysis of the platooning SoS protocol. After introducing the settings for the empirical analysis, we present the analysis results in detail.

A. Empirical Study Design

The target software of this analysis is the platooning management protocol provided by VENTOS. We utilized the StarPlateS framework to efficiently generate random scenarios for VENTOS execution and check whether the verification property was achieved on the execution logs. The detailed setup for the empirical study is presented in Table I. We generated a total of 6,525 scenarios and execution traces for the scenarios (42 GB). In each scenario, events representing the execution of the platooning operation, such as `Merge` or `Leave` were inserted at 20-second intervals. Thus, in 100 seconds simulation, at least five platooning operations should be executed. For reliability evaluation factors, *operation_success_rate* (*OSR*) and *collision_existence* (*COLL*) were used. The *OSR* property is provided by the StarPlateS framework, and this property is one of the conventional verification properties for cloud systems [29]. The *COLL* property is newly added to verify the existence of collisions in the simulation. Originally, the VENTOS simulator had a collision-free option in the simulation; thus, no collision occurred. We turned off the collision-free setting by following the VENTOS manual and added the collision detection module in StarPlateS. We

TABLE I: Overall setups of the empirical study

Parameter	Setting
Scenario setting	
Number of generated scenarios	6,525 scenarios
Duration between events	20 logic seconds
Verification property	
OSR threshold	0.8
COLL threshold	1
Simulation setting	
Duration of a single simulation	100 logic seconds
Number of generated platoons	2−4 platoons
Size of each platoon	2−6 vehicles
Map	An infinite length road with 3 lanes
Environmental objects	Human-Driven Vehicle (HDV) generated every 5 seconds
Vehicle setting	
Types of vehicles	Passenger, Truck
Autonomous driving policy	Krauss, ACC, CACC models
Hardware specification	
CPU	Intel i7-9700K CPU @3.60GHz
Memory	32GB
SSD	Samsung SSD 860 Pro 500GB
Software specification	
OS	Ubuntu 16.04 64-bit
OMNET++	5.4.1
JAVA version	java 1.8.0

set the threshold values of 0.8 and 1 to distinguish success from failure for the *OSR* and *COLL* properties, respectively.

A single simulation time was 100 logic seconds of the simulator. It actually takes approximately 10 to 15 seconds to execute in VENTOS. In each simulation, the number of platoons was randomly selected, from 2 to 4, with a randomly assigned size of 2 to 6. The simulation map is assigned by an infinite length of road with three lanes. The starting lanes of the platoons were randomly selected. In addition, to cover environmental uncertainties in the reliability analysis, we added human-driven vehicles (HDVs) that cannot communicate with platoon vehicles and randomly change the speed and lanes in the simulation. The HDVs are generated every 5 seconds; thus, approximately 20 HDVs are generated in a single simulation.

To cover the diversity and heterogeneity of platooning SoS, we added two types of vehicles, passenger and trucks, as platooning vehicles and HDVs. Every vehicle had an autonomous driving model out of the Krauss, adaptive cruise control (ACC), and cooperative ACC (CACC) models provided by SUMO[1]. In the simulation, platooning vehicles use the CACC model as the default option. However, in specific cases, such as persistent communication failures or a sudden loss of leaders by collision, the driving policy of the platooning vehicles is changed to the ACC or the Krauss model.

The hardware and software specifications for the empirical study are also described in Table I. We followed the default settings of the VENTOS installation: Ubuntu 16.04, OMNET++ 5.4.1, and Java 1.8.0. In the next section, we elucidate the empirical analysis results based on these settings.

[1]https://sumo.dlr.de/docs/

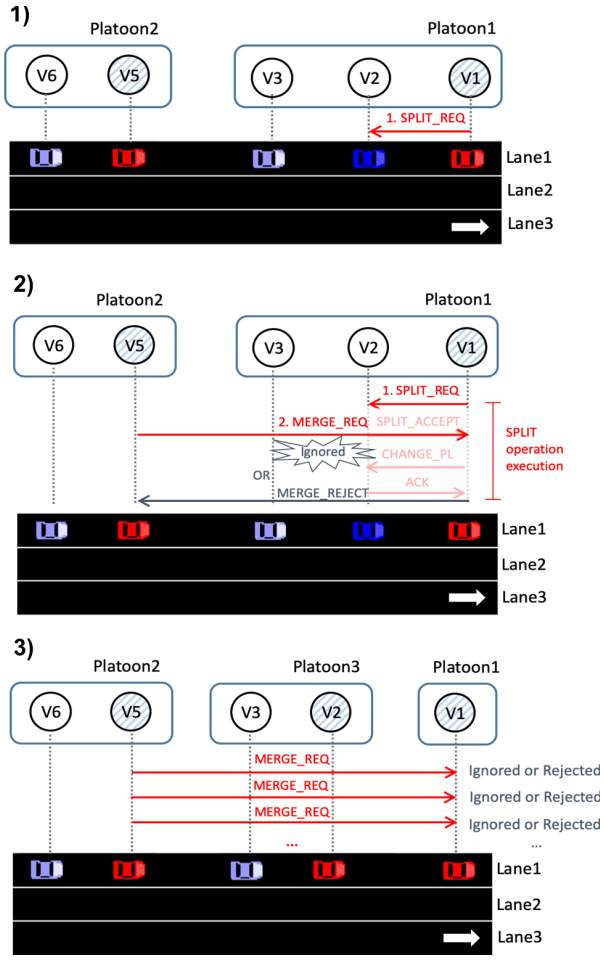


Fig. 3: Illustrative example of executions of failure class 2 in *OSR* analysis

B. Operational Success Rate Analysis

Based on the settings described in the previous section, we investigated the failed execution logs using the *OSR* property. We found ten failure situations that always violate the *OSR* property. Table II elucidates the detailed fault analysis results for each situation, which are organized as failure classes. The failure scenarios for each class are described by the context, triggering event of errors, and symptoms. Furthermore, the failures are categorized into four types: Incorrect logic; Missing logic; Non-occurrence of expected events on expected time; and Communication concurrency error by their root causes and execution context. Based on the existing categorization taxonomy of interaction failures among humans and robots [30], we added new classifications, like Communication concurrency errors or Cascading failure, and applied them in the analysis. For example, the failure classes 5, 6, and 10 are caused by the "MergeRequestAttempt bug", which is an incorrect logic applied during the `MERGE_REQ` process. Similarly, failure classes 7 and 8 were caused by the missing logic of specific protocol execution, such as `LeaderLeave` and `MiddleFollowerLeave` operations. Class 9 denotes that during the `MiddleFollowerLeave`

TABLE II: Empirical analysis results on *OSR* verification property

Class ID	Failure scenario					Faults in code	Lines in code
	Name	Context	Triggering event	Symptoms	Categorization		*05_PlatoonMg.cc*
Class 1	*Simultaneous Merge & Merge*	During the Merge operation	The rear platoon leader requests Merge by OptSize policy to the same leader who simultaneously requests Merge operation.	Constantly requests Merge to the original leader	Communication concurrency error	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744
Class 2	*Simultaneous Split & Merge*	During the Split operation	The rear platoon leader requests Merge by OptSize policy to the same leader who simultaneously requests Split operation.	Constantly requests Merge to the original leader	Communication concurrency error	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744
Class 3	*Simultaneous LeaderLeave & Merge*	During the Leader Leave operation	The rear platoon leader requests Merge by OptSize policy to the same leader who simultaneously requests LeaderLeave operation.	Constantly requests Merge to the original leader	Communication concurrency error	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744
Class 4	*Simultaneous FollowerLeave & Merge*	During the Follower Leave operation	The rear platoon leader requests Merge by OptSize policy to the same leader who simultaneously requests FollowerLeave operation.	Constantly requests Merge to the original leader	Communication concurrency error	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744
Class 5	*Split Optsize*	During the Split operation	The rear platoon leader requests Merge to the newly split platoon by OptSize policy	Constantly requests Merge to the newly split platoon leader	Incorrect logic	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744
Class 6	*LeaderLeave Optsize 1*	During the Leader Leave operation	The rear platoon leader requests Merge to the new platoon leader by OptSize policy	Constantly requests Merge to the new leader	Incorrect logic	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744
Class 7	*LeaderLeave Optsize 2*	During the Leader Leave operation	The new platoon leader requests Merge to the left leader by OptSize policy	Constantly requests Merge to the left leader	Missing logic	- ChangeVehStateLastly bug OR - LeavedVehList bug	- 1397∼1424 - 726∼745
Class 8	*MiddleFollower Leave Optsize 1*	During the Middle Follower Leave operation	The intermediate platoon leader requests Merge to the left vehicle by OptSize policy	Constantly requests Merge to the left leader	Missing logic	- FollowerLeaveProtocol bug OR - LeavedVehList bug	- 1764∼1798 - 726∼745
Class 9	*MiddleFollower Leave Optsize 2*	During the Middle Follower Leave operation	The rear platoon leader requests Merge to the intermediate leader by OptSize policy	Constantly requests Merge to the intermediate leader	Non-occurrence of expected events on expected time	- FollowerLeaveProtocol bug AND - LeaveSplitCaller bug	- 1780∼1796
Class 10	*EndFollower Leave Optsize*	During the Split operation in the End Follower Leave operation	The rear platoon leader requests Merge to the left vehicle by OptSize policy	Constantly requests Merge to the left leader	Incorrect logic	- BusyReplying bug OR - MERGE_REQUEST attempt bug	- 609∼636 - 729 ∼744

operation, certain events were not correctly executed in a specific condition. Finally, classes 1 to 4 explains the concurrency failure cases, where a platoon leader requests operations, such as `LeaderLeave` or `Merge`, and is also requested to `Merge` by the other platoon leader simultaneously.

Fig. 3 illustrates the execution of failure class 2, which is a simultaneous `Split` and `Merge`. When a platoon follower, V2, requests `Split` to V1, the other platoon leader behind the platoon, V5, could simultaneously request `Merge` operation to V1. In the VENTOS protocol, the `Merge` request delivered during the `Split` execution, is ignored or rejected. However, even after the `Split` operation is completed, we observed that the rear platoon leader, V5, continuously requests `Merge` to the same vehicle, V1. This failure situation adversely affects the operation of the related platoon vehicles, V1 and V5, and may cause overall operation delays or execution failures.

The seventh and eighth columns of Table II describe the corresponding faults and their locations. For example, "`MERGE_REQUEST` attempt bug" is one where the "MergeRequestAttempts" value is changed to the initial value after the three times requests, but the sender vehicle continuously requests `Merge`. To resolve the bug, the additional data variable is necessary to check the busy vehicles. The bug is located in the code blocks of *05_platoonMg.cc* file in lines 729-744. The OR symbol in the seventh column denotes that, if one of the faults is fixed, the failure is resolved. The AND symbol indicates that both of the faults must be fixed to resolve the failure. The faults identified in this study are mostly due to the absence of several statements, which are missing logic or incorrect logic flows for unexpected situations and interactions in the protocol. This implies that most solutions for the bugs include adding new conditional code blocks followed by the improvement of the protocol model design. On the web page of the PLTBench[2], we provided detailed information for all code-level faults and their occurrence patterns with more various illustrative examples.

C. Collision Analysis

Similar to the *OSR* property-based analysis, we conducted a detailed analysis on failure cases that violated the *COLL* property. Table III describes the failure scenarios, code-level faults, and categorization of the root causes. We focused on the six types of failure classes that were caused by platooning operations. There are more types of failure scenarios that cause collisions in Table V. However, we conducted the analysis focusing on failure scenarios that have the root causes inside the platooning system. First, we confirmed that collisions could be caused by all platooning operations that are executable in the VENTOS simulator. The root cause of collisions in the majority of detected failure classes was that the platooning operation logic did not consider the presence of environmental vehicles (i.e., HDVs) or other platoon vehicles in the rear. Therefore, most of the root causes of detected collisions are the omission of the logic that considers the distance from the rear vehicle in each operation. In the case of `SpeedChange` operation, there is no code of function call in the *05_platoonMg.cc* file, but in the other code section. Similarly, in order to solve the majority of the root causes found above, specific codes must be newly added.

Fig. 4 depicts one of the interesting failure scenarios detected in the collision analysis. The illustrated failure case belongs to failure class six in Table III and is a type of comprehensive and cascading failure scenario. First, in a situation where three platoons, V1, V2, and V5, request simultaneous `Merge` to the front leader, illustrated in 1) in Fig. 4, the inter-platoon distance between V1 and V2 is reduced to perform

[2]https://sites.google.com/se.kaist.ac.kr/pltbench/

TABLE III: Empirical analysis results on *COLL* verification property

Class ID	Failure scenario					Faults in code	Lines in code
	Name	Context	Triggering event	Symptoms	Categorization		*05_PlatoonMg.cc*
Class 1	*Split*	During the Split operation	The split operation from the platoon results the platoon vehicles to increase the front distance.	Collision occurred	Missing logic	- Missing distance checking during Split	- 1381 ∼ 1393
Class 2	*LeaderLeave*	During the Leader Leave operation	The leave operation from the platoon results the platoon vehicles to increase the front distance.	Collision occurred	Missing logic	- Missing distance checking during LeaderLeave	- 1581 ∼ 1596
Class 3	*MiddleFollowerLeave*	During the Middle Follower Leave operation	The leave operation from the middle of the platoon results in a split operation.	Collision occurred	Missing logic	- Missing distance checking during MiddleFollowerLeave	- 1738 ∼ 1749 - 1764 ∼ 1777
Class 4	*EndFollowerLeave*	During the End Follower Leave operation	The end vehicle prepares to leave the platoon by increasing the front distance.	Collision occurred	Missing logic	- Missing distance checking during EndFollowerLeave	- 1738 ∼ 1749 - 1751 ∼ 1763
Class 5	*SpeedChange*	During the Speed Change operation	The vehicle changes its speed	Collision occurred	Missing logic	- Missing distance checking during SpeedChange	in vehicle driving setting code
Class 6	*Merge*	During the Merge operation	The merge operation is unsuccessful resulting in increasing the front distance.	Collision occurred	Missing logic Cascading failure	- Missing distance checking during Merge - BusyReplying bug OR - MERGE_REQUEST attempt bug	- 727 ∼745

the `Merge` operations, as illustrated in 2). However, owing to the simultaneous requests of the `Merge` operations, V2 is overloaded and the `Merge` with V1, which is already in progress, eventually fails. Due to the failure of the progressing `Merge` operation, V2 needs to increase the inter-platoon distance to the original distance. Inevitably, in this process, the V5 also slows down to maintain the inter-platoon distance with V2, and V5 collides with the rear environmental vehicle in the end. In this manner, failure situations in platooning SoS occur during concurrent and intricate interactions; thus analyzing the failures is a highly time-consuming task. We generated the benchmark dataset based on the detailed analysis results.

V. Benchmark Dataset: PLTBench

In this section, we explain the detailed procedure for generating a benchmark dataset and elucidate the components of the PLTBench in several aspects.

A. Benchmark Generation Procedure

Fig. 5 depicts the overall procedure for generating the benchmark dataset. We performed two full examinations of all failed logs. The goal of the first examination is to establish the fault knowledge base for the failures that occurred in platooning SoS. The results of the first examination are described in Section IV. We specified the failure scenarios in the form of the context, triggering events, and symptoms. We further analyzed the root causes and failure patterns of each failure class. By generating the fault knowledge base including the aforementioned information, the basis for generating a benchmark dataset was completed.

Using the fault knowledge base, we performed a labeling procedure for all failed logs in the second examination phase. First, we labeled the failed logs by checking whether the occurrence patterns for each failure class are detected. Then, we executed the failed scenarios again and confirmed that the labels were correctly assigned to the failure executions in the GUI simulation. We performed the classification based on the analysis results for *OSR* and *COLL*, respectively.

Subsequently, we analyzed and improved the quality of the generated dataset. We analyzed the data set on three aspects: *Correctness*, *Availability*, and *Balance*. To check the correctness of the data set, we assigned the failed logs to the

Fig. 4: Illustrative example of executions of failure class 6 in *COLL*

project members and cross-checked the labeling result so that every failed log could be checked by at least two people. To increase the availability of the dataset, we implemented a web page for the accessible utilization of the benchmark dataset. Finally, we investigated the balance of the dataset, which checks whether the distribution of data was not too skewed. The details of the composition of the benchmark dataset are described in the next section.

B. PLTBench Composition

The benchmark dataset is mainly composed of raw logs with scenarios, analysis results, and classification results with statistics. The whole benchmark dataset including the analysis and classification results can be found on our web page. As it is described, the raw logs are approximately 42 GB of simulation execution traces, and consist of vehicle location data, emission data, platooning configuration data, and platooning communication data. Each datapoint was stored in units of 0.5 milliseconds. Additionally, we gathered console messages from VENTOS simulator for each simulation and saved them. The console logs contain information on the state changes of platooning vehicles and collision messages. To provide the

Fig. 5: Overall process of generating the benchmark dataset for platooning SoS

reproducibility of the generated logs, we also added the scenarios and configurations of the logs in the dataset. The initial vehicle configurations and scenarios with platooning operation execution at specific times were included. Therefore, users can reproduce the execution traces or check the simulation in GUI by using the scenarios and configurations.

C. Statistics of the Dataset

In addition to the detailed empirical analysis results for the reliability properties described in Section IV, we provide failure scenario classification results and statistics for all failed logs. The classification results for failure scenarios that violate the *OSR* property are listed in Table IV. We found 3,256 numbers of failed logs and 3,830 cases of failure executions. The difference between the total number of failed logs and the number of actual detected failure classes appears because multiple failure scenarios occur simultaneously in a single log. On average, it was confirmed that 1.17 failure classes were found in one log.

TABLE IV: *OSR* analysis statistics

Class ID	**Counts**
Class 1 (Simultaneous `Merge` & `Merge`)	213
Class 2 (Simultaneous `Split` & `Merge`)	62
Class 3 (Simultaneous `LeaderLeave` & `Merge`)	133
Class 4 (Simultaneous `FollowerLeave` & `Merge`)	138
Class 5 (`Split` optsize)	794
Class 6 (`LeaderLeave` optsize 1)	159
Class 7 (`LeaderLeave` optsize 2)	579
Class 8 (`MiddleFollowerLeave` optsize 1)	389
Class 9 (`MiddleFollowerLeave` optsize 2)	1104
Class 10 (`EndFollowerLeave` optsize)	259
Total	3830

TABLE V: *COLL* analysis statistics

Class ID	***Env_Env***	***Plt_Plt***	***Plt_Env***	**Sum**
By Env	521	9	218	748
By Plt Op	29	–	172	201
- Class 1 (`Split`)	11	–	78	89
- Class 2 (`LeaderLeave`)	6	–	18	24
- Class 3 (`MiddleFollowerLeave`)	3	–	26	29
- Class 4 (`EndFollowerLeave`)	1	–	25	26
- Class 5 (`SpeedChange`)	8	–	23	31
- Class 6 (`Merge`)	0	–	2	2
Unknown	4	–	12	16
Total	554	9	402	965

It is also observed that the data distribution is affected according to the failure classes. Generally, the simultaneous failure classes were smaller than the other classes. This trend is caused by the generation of random scenarios. Scenarios in which specific operations are executed simultaneously are less likely to be randomly generated than scenarios that do not involve simultaneous operation executions. Therefore, differences in distribution are inevitable because of the difficulties in generating edge cases in the process of generating random scenarios. From the viewpoint of data balancing, uniformly distributed data is not always the best option for a dataset [31]. In the credit card defrauded dataset, only 3.9% of the data are related to the fraud [32], and only 0.4% is positive in the HIV prevalence data set [31]. Nevertheless, we plan to generate more scenarios involving simultaneous operation execution. We will use a guided method by modifying the random scenario generation module in StarPlateS and provide more numbers of the failure executions corresponding to the simultaneous operation executions.

Table V describes the statistics of the classification results by the failure classes that violate the *COLL* property. We conducted the classification process according to the collision subjects: “Env vehicle (i.e., HDV) with Plt vehicle”, “Env vehicle with Env vehicle”, and “Plt vehicle with Plt vehicle” and to the cause of collisions: “by Env vehicle”, “*By Plt Op*erations(e.g., `Merge`, `MiddleFollowerLeave`), and “by unknown”. The simultaneous occurrence of multiple failure classes showed a similar trend in collision failures. We identified 965 failure cases among the 900 failed logs. However, we found that the logs having multiple failure cases

contain more numbers of failure cases at once. For instance, the maximum number of multiple failure cases in a single failed log is six in the *COLL* analysis result.

The failure cases we mainly focus on are collisions caused by platooning operations (*By Plt Op*) or collisions involving platoon vehicles (*Plt_Plt* and *Plt_Env*). All collision cases among platoon vehicles (*Plt_Plt*) are caused by HDVs, which suddenly change their speed or driving lanes just in front of truck platoon vehicles. Most crashes caused by environmental HDVs (*By Env*) have similar failure scenarios to sudden lane change and speed change. In the collision cases caused by the platooning operations (*By Plt Op*), we found 172 cases in total. The most common case is by `Split` operation. This is because the `Split` operation is called during the execution of all `Leave` operations. The collision scenarios are described in detail in Section IV-C.

The *Unknown* cases are the scenarios that are difficult to reproduce or the cases in which the causes of the collisions are unclear by checking the simulation. For instance, one example of an *Unknown* case is a situation in which an HDV increases the distance to its rear vehicle corresponding to a platoon vehicle in the process of `Split` in the other lane. To analyze the unknown cases, we plan to perform the deep code-level analysis for all simulator codes, as well as the platooning protocol codes in VENTOS.

D. PLTBench Web Page

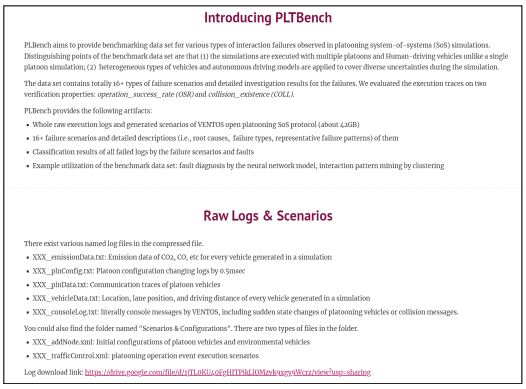
Fig. 6 depicts the example contents on our web page. We established a web page for PLTBench to maximize the accessibility and availability of the dataset for users. All data and descriptions related to the PLTBench are listed on the web page. Users can easily download the raw logs and classification results based on failure classes. If users want to reproduce the failed execution logs, they can execute the provided scenarios in VENTOS and check the results by the simulation. Fig. 6a shows the basic introduction of PLTBench and the description of raw logs and scenarios with the download link. Fig. 6b illustrates the detailed empirical analysis results on the web page. The classification results of the *COLL* failure classes are shown in Fig. 6c. Finally, examples of ways to utilize the PLTBench were elucidated, as shown in Fig. 6d. The details of the example uses of the benchmark dataset are explained in the next section.

VI. Application Examples of Benchmark Dataset

Herein, we introduce two experiments that used the PLTBench dataset. The first experiment evaluated the performance of a neural network-based fault diagnosis technique for platooning SoS failures. The second result depicts the precision of clustering results, where the clustering technique aims to extract failure patterns from logs and to help to establish fault knowledge of the target system.

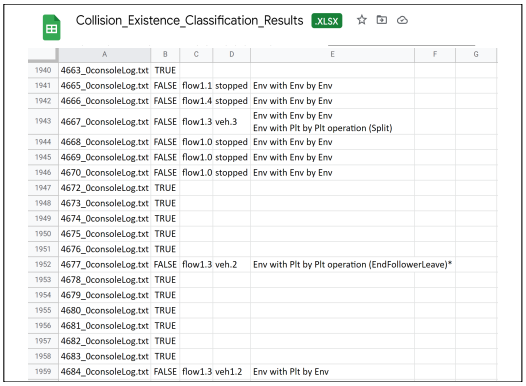
A. Log-Based Fault Diagnosis Techniques

Safety-critical systems or social infrastructure systems require immediate responses when failures occur. One of the

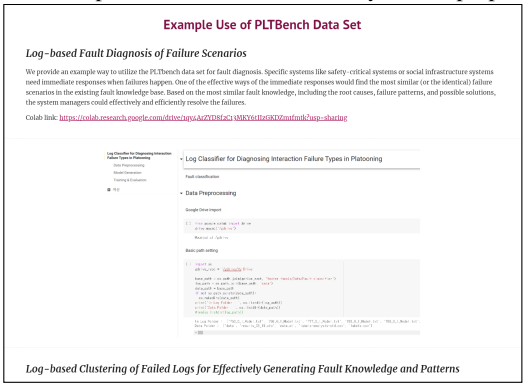


a Basic description and download link of the raw logs

b Detailed analysis result and the classification file link



c An example of classification result by *COLL* property



d Example description of utilizing the benchmark data set

Fig. 6: Example arrangement at the actual benchmark web page

TABLE VI: Evaluation results of the fault diagnosis technique on the PLTBench data set

Evaluation method	Avg. of best accuracies	Total best accuracy
six-fold cross-validation	0.9806	0.9962

effective ways of the immediate responses is to find the most similar (or the identical) failure scenarios in the existing fault knowledge base. Based on the most similar fault knowledge, including the root causes, failure patterns, and possible solutions, system managers could efficiently resolve the failures that occurred. Table VI describes the evaluation results of the neural network-based fault diagnosis technique. We built a three-layered neural network and added a pre-processing module for the platooning SoS logs. We performed a six-fold cross-validation on the technique. The technique achieved 0.9806 for the average of the best accuracy values in each fold and 0.9962 for the total best accuracy value. Detailed codes and evaluation metrics are provided on the PLTBench web page, as shown in Fig. 6d. Using this PLTBench dataset, we showed that it is possible to evaluate and analyze the performance of a log-based fault diagnosis approach.

B. Log-Based Faulty Pattern Mining Techniques

Several fault analysis techniques, such as fault diagnosis techniques, require a detailed fault knowledge base of the target system [29], [33], [34] for using the techniques. This implies that several fault analysis or detection techniques need fault data, failure scenarios, or patterns that have already been analyzed and classified. However, generating such a fault knowledge base requires considerable effort. Log-based clustering technique could provide the basis for generating a fault knowledge base by mining common failure patterns. Because the PLTBench provides a completely labeled dataset for the failures in platooning SoS, the dataset could be utilized to evaluate such unsupervised clustering techniques.

We evaluated two log-based clustering techniques and the F1P evaluation metric for overlapping clustering evaluation, which are open to the StarPlateS repository[3]. Fig. 7 illustrates the evaluation results of two log-based failure scenario clustering techniques: 0.557 value for Base LCS and 0.796 value for TIME LCS. Here, LCS denotes the longest common subsequence. The goal of the two techniques is to mine accurate failure patterns by clustering similar failed logs. In this evaluation process, we utilized the classification results of failed logs that violate the *OSR* property. This evaluation can be performed only when the completely labeled failed logs are provided. Through the PLTBench dataset, we carried out the precision comparison of the two clustering techniques.

VII. Conclusion

We presented an open benchmark dataset for platooning SoS, PLTBench, which covers internal and external uncertainty factors by generating heterogeneous types of platoon vehicles and human-driven environmental vehicles in simulations. Existing testing or verification techniques did not fully cover the uncertainty factors during the testing or verification process and were limited to a single and homogeneous platoon simulation, without environmental objects. We addressed the limitations and proposed a PLTBench dataset consisting of 42 GB of raw simulation logs and scenarios, detailed empirical analysis results on failures and faults based on the two reliability attributes, classification results of all failed logs by the analyzed failures, and example utilization of the dataset. We also built a web page to increase the accessibility of the artifacts and descriptions for the dataset. We expect that the PLTBench dataset could enrich the typical failure scenarios and experimental data set for future studies targeting the platooning SoS.

We plan to improve the benchmark dataset in two ways. First is by providing solution codes for the detected faults in the VENTOS platooning protocol codes. We sent the detected faults and possible solutions to the managers of VENTOS, but did not get a response yet. If the owners of VENTOS agree on the patches, we can directly add the solutions to our web page. We expect that the patch codes can be utilized in diverse research studies. Further, we plan to generate more scenarios for simultaneous operation execution classes to improve the balance of the distribution of the dataset.

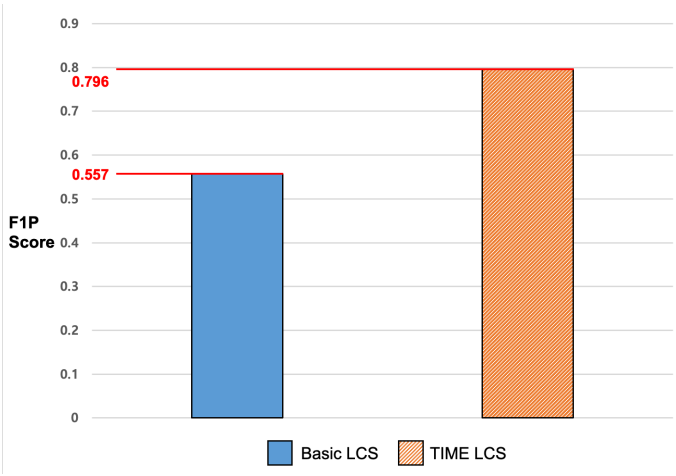


Fig. 7: *F1P* evaluation results of log-based clustering techniques: Base LCS and TIME LCS

VIII. Acknowledgement

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2020-0-01795) and (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation).

References

[1] J. Janai, F. Güney, A. Behl, A. Geiger, et al., "Computer vision for autonomous vehicles: Problems, datasets and state of the art," Founda-

[3]https://github.com/KAIST-SE-Lab/StarPlateS

tions and Trends® in Computer Graphics and Vision, vol. 12, no. 1–3, pp. 1–308, 2020.
[2] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," Annual Review of Control, Robotics, and Autonomous Systems, vol. 1, pp. 187–210, 2018.
[3] G. Li, Y. Yang, T. Zhang, X. Qu, D. Cao, B. Cheng, and K. Li, "Risk assessment based collision avoidance decision-making for autonomous vehicles in multi-scenarios," Transportation research part C: emerging technologies, vol. 122, p. 102820, 2021.
[4] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in 2018 26th Telecommunications Forum (TELFOR), pp. 420–425, IEEE, 2018.
[5] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 174–179, IEEE, 2017.
[6] C. Pek, S. Manzinger, M. Koschi, and M. Althoff, "Using online verification to prevent autonomous vehicles from causing accidents," Nature Machine Intelligence, vol. 2, no. 9, pp. 518–528, 2020.
[7] F. Gruber and M. Althoff, "Anytime safety verification of autonomous vehicles," in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp. 1708–1714, IEEE, 2018.
[8] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 1–7, IEEE, 2018.
[9] X. Zhao, V. Robu, D. Flynn, K. Salako, and L. Strigini, "Assessing the safety and reliability of autonomous vehicles from road testing," in 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), pp. 13–23, IEEE, 2019.
[10] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," Journal of Systems and Software, vol. 137, pp. 197–215, 2018.
[11] M. Wang, S. van Maarseveen, R. Happee, O. Tool, and B. van Arem, "Benefits and risks of truck platooning on freeway operations near entrance ramp," Transportation Research Record, vol. 2673, no. 8, pp. 588–602, 2019.
[12] Y. Jo, J. Kim, C. Oh, I. Kim, and G. Lee, "Benefits of travel time savings by truck platooning in korean freeway networks," Transport Policy, vol. 83, pp. 37–45, 2019.
[13] B. Jacob and O. A. de Chalendar, "Truck platooning: Expected benefits and implementation conditions on highways," in Heavy Vehicle Transportant Technology (HVTT) International Symposium, 2018.
[14] H. Hexmoor and R. Alshiddi, "Salient benefits of platooning," EPiC Series in Computing, vol. 75, pp. 49–58, 2021.
[15] K. Salari and J. Ortega, "Experimental investigation of the aerodynamic benefits of truck platooning," tech. rep., SAE Technical Paper, 2018.
[16] J. Axelsson, "Business models and roles for mediating services in a truck platooning system-of-systems," in 2019 14th Annual Conference System of Systems Engineering (SoSE), pp. 113–118, IEEE, 2019.
[17] S. Hyun, J. Song, S. Shin, and D.-H. Bae, "Statistical verification framework for platooning system of systems with uncertainty," in 2019 26th Asia-Pacific Software Engineering Conference (APSEC), pp. 212–219, IEEE, 2019.
[18] Y.-J. Shin, L. Liu, S. Hyun, and D.-H. Bae, "Platooning legos: An open physical exemplar for engineering self-adaptive cyber-physical systems-of-systems," in 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 231–237, IEEE, 2021.
[19] A. K. Saberi, E. Barbier, F. Benders, and M. Van Den Brand, "On functional safety methods: A system of systems approach," in 2018 Annual IEEE International Systems Conference (SysCon), pp. 1–6, IEEE, 2018.
[20] M. Kamali, S. Linker, and M. Fisher, "Modular Verification of Vehicle Platooning with respect to Decisions, Space and Time," in International Workshop on Formal Techniques for Safety-Critical Systems, pp. 18–36, Springer, 2018.
[21] M. Elgharbawy, "A Big Testing Framework for Automated Truck Driving," Urban transportation and construction, vol. 4, no. 1, pp. e27–e27, 2019.
[22] K. Meinke, "Learning-based Testing of Cyber-Physical Systems-of-Systems: A Platooning Study," in European Workshop on Performance Engineering, pp. 135–151, Springer, 2017.
[23] M. Amoozadeh, H. Deng, C.-N. Chuah, H. M. Zhang, and D. Ghosal, "Platoon Management with Cooperative Adaptive Cruise Control Enabled by VANET," Vehicular communications, vol. 2, no. 2, pp. 110–123, 2015.
[24] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and ..., 2008.
[25] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp. 2575–2582, IEEE, 2018.
[26] E. Vasconcelos Filho, N. Guedes, B. Vieira, M. Mestre, R. Severino, B. Gonçalves, A. Koubaa, and E. Tovar, "Towards a cooperative robotic platooning testbed," in 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 332–337, IEEE, 2020.
[27] B. Vieira, R. Severino, A. Koubâa, and E. Tovar, "Towards a realistic simulation framework for vehicular platooning applications," in 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), pp. 93–94, IEEE, 2019.
[28] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a Nutshell," International Journal on Software Tools for Technology Transfer (STTT), vol. 1, no. 1, pp. 134–152, 1997.
[29] C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. D. S. Silvestre, "Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned," Journal of Systems and Software, vol. 139, pp. 84–106, 2018.
[30] D. J. Brooks, A human-centric approach to autonomous robot failures. PhD thesis, University of Massachusetts Lowell, 2017.
[31] M. Y. Arafat, S. Hoque, S. Xu, and D. M. Farid, "Machine learning for mining imbalanced data," IAENG International Journal of Computer Science, vol. 46, no. 2, pp. 332–348, 2019.
[32] G. Figueroa, Y.-S. Chen, N. Avila, and C.-C. Chu, "Improved practices in machine learning algorithms for ntl detection with imbalanced data," in 2017 IEEE Power & Energy Society General Meeting, pp. 1–5, IEEE, 2017.
[33] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1285–1298, 2017.
[34] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, et al., "Robust log-based anomaly detection on unstable log data," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 807–817, 2019.