Representative Dataset Generation Framework for AI-based Failure Analysis during real-time Validation of Automotive Software Systems

Mohammad Abboush
Technische Universität Clausthal, Institute for Software and Systems Engineering
mohammad.abboush@tu-clausthal.de

Christoph Knieke
Technische Universität Clausthal, Institute for Software and Systems Engineering
christoph.knieke@tu-clausthal.de

Andreas Rausch
Technische Universität Clausthal, Institute for Software and Systems Engineering
andreas.rausch@tu-clausthal.de

Abstract

Recently, thanks to its ability of extracting knowledge from historical data, the data-driven approach has been widely used in various phases of the system development life cycle. In real-time system validation, remarkable achievements have been accomplished in developing an intelligent failure analysis based on historical data. However, despite its superiority over other conventional approaches, e.g., model-based and signal-based, the availability of representative datasets persists as a major challenge. Thus, for different engineering applications, new solutions to generate representative faulty data in different forms should be explored. Therefore, in this study, a novel approach based on Hardware-in-the-Loop (HIL) simulation and real-time Fault Injection (FI) method is proposed to generate and collect data samples under single and simultaneous faults for Machine Learning (ML) applications during system validation phases. The developed framework can generate not only sequential data, but also textual data including fault logs. The results show the applicability of the proposed framework in simulating and capturing the system behaviour under faults within the system components.

Keywords: HIL testing, fault injection, automotive software systems development, failure analysis, machine learning.

1. Introduction

In accordance with model-based design and the V-model development approach, the System Under Test (SUT) is tested at each level of the development process. In other words, the testing and validation process are performed for different states of the SUT, i.e., the executable model, the model code and the implemented code on the host and target machine (Garousi et al., 2018). Several platforms have been leveraged to carry out the testing and validation activities, which are known as X-in-the-Loop (Shokry and Hinchey, 2009). As such, Hardware-in-the-Loop (HIL) real-time simulation is recommended by ISO 26262 as a safe, flexible, reliable, and effective platform (Himmler et al., 2012). Recently, HIL has played a vital role in the verification and validation of automotive embedded control systems under time constraints. Substituting the real hardware elements with fidelity simulated system provides not only avoidance of potential risks but also reductions in testing costs for various applications e.g., electric drives, power electronics, power grids, railways and automotive (Mihalič et al., 2022). HIL-based system validation is currently a hot topic in academia and industry due to its ability to provide efficient, fast, and realistic simulations in real-time with high accuracy. On the top of that, by serving as a digital test drive platform, HIL has contributed to overcome the limitations of real test drives in terms of time, cost, effort and risk to the tester (Chen et al., 2018). However, at system integration testing level, an enormous amount of datasets from heterogeneous components and subsystems are generated as a result of tests' execution (Jordan et al., 2020). Besides, despite test automation of Test Cases (TCs), many failed TCs are documented in the test report as pass/fail (Vermeulen, 2008). Therefore, analysing the vast amount of test records based on traditional approaches is costly, difficult, and time-consuming (Nair and Koustubh, 2017). This is why an intelligent system capable of analysing the test results of HIL in an efficient way without domain knowledge is necessary.

URI: https://hdl.handle.net/10125/107263
978-0-9981331-7-1

In the last decade, the advances in the computational resources paved the way for the widespread application of data-driven approach to tackle the problem of Fault Detection and Diagnosis (FDD). Central to this approach is the idea of extracting knowledge from historical data and constructing non-linear relationships between input and output classes to discover hidden patterns. Compared to other FDD approaches, neither does it require expert knowledge nor a precise mathematical model. Hence, the data-driven approach has attracted the attention of researchers, and the development of FDD methods has increased rapidly in various technical fields. Deep Learning (DL) and Machine Learning (ML)-based methods, as subsets of the data-driven approach, are widely used in various phases of the software development life cycle, i.e., software requirements, software architecture and design, software implementation, software quality and analysis, and software maintenance (Shafiq et al., 2021). In the testing and analysis phase, for example, DL and ML tackle not only the generation and selection of TCs, but also to the detection and analysis of defects and anomalies. However, despite the remarkable accomplishments in different domains, one of the major challenges impeding DL and ML methods is the lack of datasets (Badihi et al., 2022; Neupane and Seok, 2020). As a result, the availability of datasets imposes constraints on the development process of DL-based FDD in real-world applications. For that reason, generating representative datasets, such as document logs and signal data, should be further explored.

As previously mentioned, one of the core elements of developing FDD model is the dataset. In principle, obtaining fault-free data can be realised by logging the system behaviour in fault-free mode (S. Zhang et al., 2020). However, acquiring representative data that capture the reactions of the system under faulty conditions is complicated in terms of difficulty and cost (T. Zhang et al., 2022). In automotive domain, due to confidentiality of test data, it is rare to have public real-world data containing faulty behaviour (Theissler et al., 2021). Besides, even in case of limited data existence, the ratio of faulty to healthy data is small and unstable, which in turn leads to the problem of imbalanced data. A reason for this is the difficulty in simulating the open set of failure modes under complex working conditions (Kukkala et al., 2018). Besides, capturing hardware faults is not feasible in real-world applications (Fernandes et al., 2022).

In order to train a robust intelligent FDD model, the following requirements on the dataset should be ensured: 1. a high quality dataset with a sufficient number of samples, including healthy and faulty operating conditions, 2. a high degree of fault classes coverage, 3. consideration of the single and simultaneous faults occurrence, 4. consideration of real-time constraints on system behaviour under normal and faulty conditions. 5. offering different types of datasets, i.e., time series and text data.

Therefore, tackling the problem of representative dataset is still an open concern, and collecting faulty dataset that meets the above requirements should be further explored. In this paper, we attempt to bridge this gap by proposing a novel framework for generating and collecting representative data including healthy and faulty samples during Automotive Software Systems (ASSs) development. To demonstrate the benefits and applicability of the proposed framework, a high-fidelity simulation of a gasoline engine system is used as a case study. Vehicle dynamics, environment, driver and powertrain models have also been considered to capture the detailed characteristics of the vehicle during data acquisition. The main contributions of the proposed framework can be summarised as follows:

- We propose a novel framework capable of generating real-time faulty datasets, taking into account the time constraints of the system behaviour.
- The framework can cover major types of random hardware sensor and actuator faults in ASSs.
- Not only the single faults but also the simultaneous occurrence of faults can be simulated, providing a high coverage.
- Real-time HIL simulation and high fidelity simulation models are employed providing high quality real-time coverage of the collected data.
- Finally, our proposed framework provides the ability to collect two types of representative datasets, i.e., time series and text log data.

The rest of the paper is structured as follows: Related work and other contributions are presented in Section 2. Section 3 introduces the proposed approach, highlighting the key stages of dataset collection and preparation. The implementation steps and the case study are described in Section 4 . Section 5 summarises the results and findings. Finally, Section 6 presents the conclusion and future work.

2. Related work

To tackle the problem of faulty data non-availability, several solutions have been proposed in the literature. Some of them have used online public datasets, while others had to generate the data depending on the

real prototype or simulated system. For example, (Rengasamy et al., 2020) have used a standard dataset of a gas turbine engine provided by NASA. The mentioned data is used to develop a DL-based model for prediction and diagnosis tasks. Similarly, based on Audi's industrial dataset, (Safavi et al., 2021) developed an intelligent model relying on DL methods for fault detection, isolation, identification and prediction. The target system in the work is automotive autonomous driving. Despite the advantage of employing published standard data, the fault classes are limited depending on the source, lacking the flexibility to extend the data under the same operating conditions. A static injection of the faulty sample into the data, based on normal distributions with one standard deviation, has been used as an alternative solution. Employing a real prototype has been an alternative solution for other researchers. For example, aiming at developing a robust FDD for vehicle engines, (Jung, 2020) employed a real engine to capture the system behaviour under normal and abnormal conditions. Since the data collected exactly matched the real industrial application data, the developed model showed a high degree of applicability and robustness to the environmental conditions, e.g., noise. However, the major limitation of the applied technique is the potential risk and damage to the target physical system in case of Fault Injection (FI). In the same respect, but for different applications, (Bafroui and Ohadi, 2014) have proposed a FDD model for automotive gearboxes. The developed model was developed based on data collected from an experimental test bench in the laboratory. Some other studies have been conducted based on data collected from real vehicle prototypes to develop an intelligent solution for the detection of unknown faults in the test drive records (Theissler, 2017). However, beside the high cost, this type of experimentation could pose a risk not only to the vehicle, but also to the tester in case of a critical situation. Moreover, some types of faults cannot be explored, in which the whole physical system can be damaged.

To overcome the limitations of using real physical systems, a simulation platform has been used to generate representative datasets. Based on the modelling and simulation techniques, many researchers have proposed solutions to address the problem of availability data. For example, (Tagawa et al., 2015) have employed the FI method to generate the faulty data based on the simulated system in the MATLAB/Simulink environment. In this work, four driving scenarios were used to generate faulty data along with normal data. Similarly, (Biddle and Fallah, 2021) approached the problem of faulty data non-availability by injecting five different types of faults into the simulated system, namely erratic, hardover, drift, spike and stuck-at fault. To this end, the failure modes were modelled in the simulation environment, i.e., MATLAB/IPG CarMaker co-simulation, and then the faults were artificially injected according to the fault parameters. Consequently, and based on the collected data, ML-based architecture for multi-fault in multi-sensor FDD was constructed. However, despite the ability to reproduce the test under critical conditions with a high degree of confidence, real-time constraints remain ignored. The effect of this is that the application of the target model in the real world becomes increasingly restricted. Not only that, the current developed FDD models are validated based on simulation data generated by a simulation platform. By doing so, the real industrial conditions, such as influences, noise and uncertainties, are ignored (Yin et al., 2022). Lately, with the aim of overcoming the limitations of pure simulation, HIL real-time simulation has been introduced as a solution for generating data covering various critical conditions (Gonzalez-Jimenez et al., 2021). Employing Dataset generated from HIL, (Abboush et al., 2022b) have proposed FDC model based on hybrid DL techniques to be used during the development phases of ASSs, i.e., integration testing. The basis of the developed model is the faulty data collected by programmatically injecting nine different single sensor faults into the target system in real-time without changing the model. As an extension of the mentioned work, in this study, the simultaneous occurrence of the faults are considered, including transient and permanent occurrence. Besides, the textual nature of logs from test execution results are collected for Root Causes Analysis (RCA) problem.

3. Methodology

In this section, the proposed framework is presented, including real-time HIL simulation, data analysis and management, and FI, as shown in Figure 1.

The HIL system is the core of the framework, in which the complex target system is simulated and executed in real-time. It consists of two main parts, namely the HIL simulator and the target prototype (controller). HIL simulator is responsible for executing the controlled system (plant) in real-time. Precisely, in our case the controlled system is the entire vehicle model. Engine model, dynamic vehicle model, environmental model, traffic model and powertrain model are the main subsystems of the selected system. Noteworthy, the SoftECU, i.e., virtual ECU, model is also included in the controlled system model. By doing so, the whole vehicle model including the controller

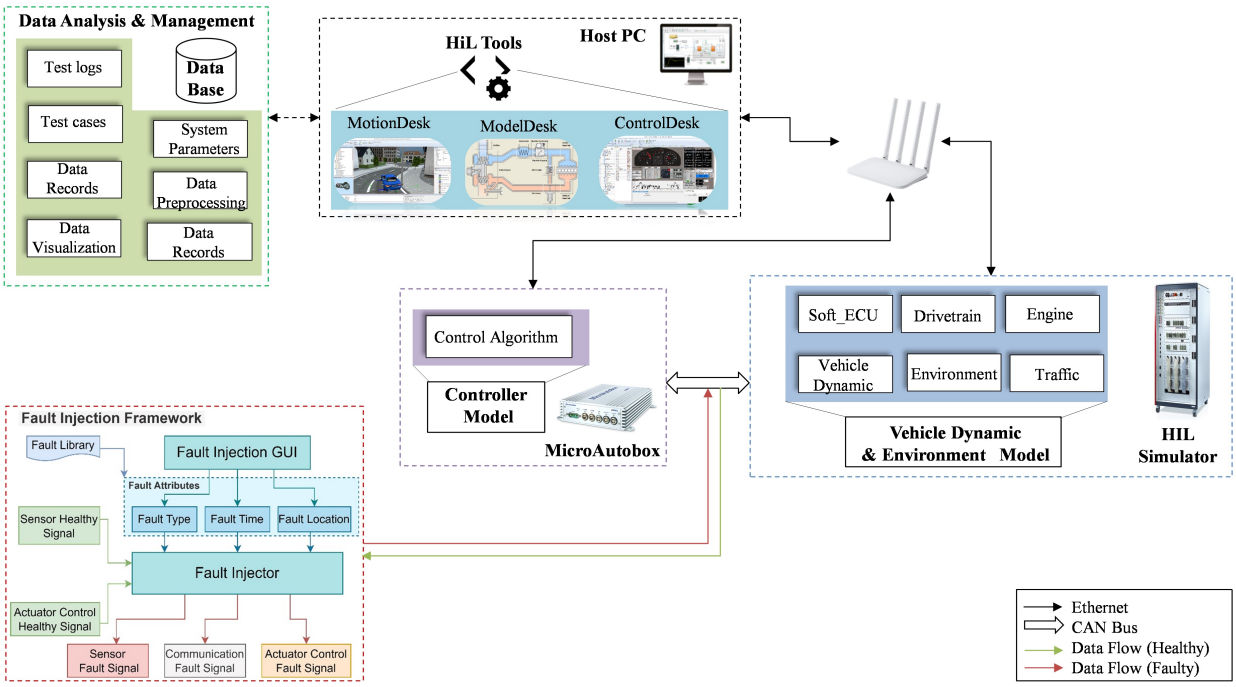


Figure 1. Proposed framework for representative dataset generation.

model can be executed in the HIL simulator, which acts as a Rapid Control Prototype (RCP), known as offline mode. On the other hand, MicroAutoBox II is the target machine of the SUT where the ECU model is deployed and executed. MicroAutoBox II is considered as RCP and acts as the real ECU. Both parts of the HIL system are connected via the CAN bus. To establish the connection, the signal interface is modelled in the simulation environment on both sides, i.e., in the ECU model and in the plant model. The generated code of the mentioned models is injected from the host PC via Ethernet into the control unit and the HIL simulator,respectively.

On the host PC, the configurations and experiments are carried out. For this purpose, four software tools from dSPACE are used, namely MotionDesk, ConfigurationDesk, ModelDesk and ControlDesk (dSPACE, 2023). In addition to the parameterisation of the system model, ModelDesk is used to design the test drive scenarios and the TCs. Thanks to the model-based design approach, once the model is configured, the code of the both models, i.e., SUT and plant, is automatically generated and deployed using ConfigurationDesk. To represent the driving environment and the dynamic traffic as a 3D visualisation, MotionDesk is employed. Finally, the instrumentation, measurements, dataset acquisition and the control of the experiments during real-time execution are performed using ControlDesk. The mentioned tool enables also switching between the offline execution mode (SoftECU) and the online execution mode (Real ECU).

FI process takes place at the signal interface between the control unit and the HIL simulator during real-time execution. By doing so, the execution of the system model of the ECU and the plant can be ensured in real-time as a black box without modification. Furthermore, the injection process is executed programmatically without extending the target system model with additional components. The input of the FI framework is healthy data representing the system behaviour under normal conditions. The target signals are then manipulated according to the fault attributes configured by the user. There are three attributes in the fault injector that should be configured before injection, i.e., fault type, fault location and FI time. Fault type covers most sensor's and actuator's malfunctions such as gain, offset, hard over, stuck-at, delay, noise, packet loss, drift and spike fault. The target sensor and actuator signal is identified as a location for faults to be injected. Duration of the FI, and the time at which the fault is injected are specified according to the driving cycle or standard system behaviour.

Finally, the data analysis and management is carried out using host PC. Here, the system requirements, the TCs specifications and the scenarios descriptions are documented. Besides, in this phase the collected

healthy and faulty datasets are stored as a result of the test execution. Two types of data are collected, sequential signal data and textual log data. This allows the collected data to be used for various tasks and applications, e.g. fault detection, diagnosis, prognosis and RCA of failed TCs. In addition to data generation, pre-processing of the data takes place so that outliers and redundant data are removed. Beyond that, the data is converted into a suitable format, e.g. CSV, MAT or XML, to be ready for the ML/DL model development process. Finally, the pre-processed data is visualised on the host PC to get a deeper understanding of the patterns and features before training the DL/ML model.

4. Case study and Experimental Implementation

To validate the applicability of the proposed framework, this section presents the selected case study, highlighting the system architecture and implementation steps.

4.1. Case study

As a case study from the automotive domain, gasoline engine system from dSPACE (dSPACE, 2023) was used to validate the proposed approach. The architecture of the system is shown in Figure 2. What can be seen is that the various systems and subsystems have been modelled to accurately reflect the actual physical characteristics. MATLAB/Simulink was selected as the graphical simulation and modelling environment to model the dynamic system. Simulink tool has significant features in designing, analysing and testing the dynamic system using the model-based approach. Besides, it offers the possibility to test the SUT in a simulated environment and automatically generate the code before deployment, supporting real-time simulation. However, despite the realistic simulation of the selected system, it does not enable driving with lateral control. To overcome this limitation, in our case , a complex gasoline engine model was integrated into the dynamic system model of the vehicle. Thereby, lateral and longitudinal driving with manual and automatic transmission can be simulated. The major subsystem models that structure the gasoline engine are exhaust, fuel, air path, cooler and the piston engine system. Various components have been modelled and interconnected in a block diagram environment, so that the comprehensive functions of the gasoline engine are simulated. Additionally, the powertrain, driver and environmental models are included to capture the overall dynamic characteristics of the vehicle. The gasoline engine is controlled by the ECU in two modes, i.e., offline and online. In offline mode, the soft ECU is internally connected to the engine, whereas a separate ECU model in the RCP is used as the real ECU to enable online mode. Finally, the interface between the real ECU model and the gasoline engine model is realised using a real-time interface CAN multimessage blockset (RTICANMM) wherein the fault injections are configured and modelled. Thanks to the feature of RTICANMM, the system signals can be accessed and manipulated programmatically without changing the original system model.

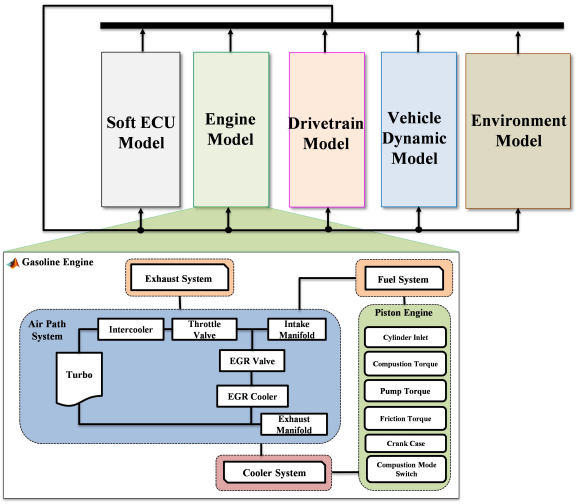


Figure 2. System architecture of the case study.

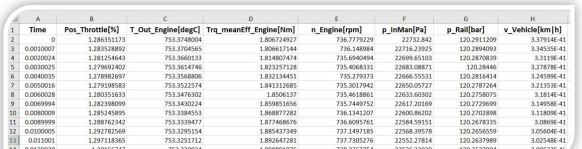
4.2. Experimental setup

There are three levels of configurations in our case study, namely system model configurations, experiments' configurations and FI configurations. At the first level, dSPACE's software tools, i.e., ModelDesk and ConfigurationDesk, are used to specify the parameters and variables of the models based on the GUI. Additionally, driving scenarios and TCs are designed based on the specifications. In our case, two driving scenarios are selected, namely "Highway" and "Ftp 75". By doing so, driving scenarios on the highway, on non-urban/open roads and in urban traffic can be covered. Once the system and test configurations are specified, the code can be automatically generated and deployed to the target machines using ConfigurationDesk. Two target machines are employed in our study, MicroAutoBox II for the ECU system and the HIL SCALEXIO simulator for the entire controlled vehicle system. Finally, the 3D visualisation of the driving environment, i.e., the defined scenarios, is modelled with MotionDesk.

On the second level, the configurations of the fault

injector are set. To be specific, the FI attributes,i.e., fault location, fault type and FI time, are configured. As already mentioned, all system signals, i.e., sensors and actuators, can be accessed via the CAN bus interface. Therefore, the degree of coverage of our developed framework is high compared to the traditional approach. Some examples of sensors signals accessed are crank angle, battery voltage, engine speed, accelerator pedal position (APP), ignition and starter request, EGR mass flow, intake and exhaust manifold pressure, fuel pressure, coolant temperature and railbar sensor. The actuators involved are the throttle valve, the fuel metering unit, the pressure control valve and the ignition angle adjuster. Due to the significant effect of a fault in the APP and engine speed sensor on the vehicle behaviour, we selected these locations for FI in our study. On the other hand, nine different types of the single faults and 15 combinations have been identified to demonstrate the coverage of the fault data generation. By this way, not only the single fault-based dataset but also the concurrent faults-based dataset can be collected. Finally, when to inject the fault and the duration of the injection are determined based on the selected driving cycle. FI can be permanent until the faulty component is treated, or it can be temporary occurring frequently for a specific duration. By injecting permanent and transient faults, both balanced and imbalanced datasets can be acquired for developing a reliable FDD model for real-world applications. The detailed configuration of the FI framework for single and simultaneous faults is shown in (Abboush et al., 2022a).

The experimental setup takes place at the last level, where the measurements, recording and instrumentation are carried out. The sampling time of the signal measurements is set to 0.001 sec in this study. At the system level, 6 variables of the target system are selected to represent the recorded system behaviour under normal and faulty conditions. These system variables are throttle position, engine temperature, mean effective engine torque, engine speed, intake manifold pressure, rail pressure and vehicle speed, as illustrated in Figure 3.



	A	B	C	D	E	F	G	H
1	Time	Pos_Throttle[%]	T_Out_Engine[degC]	Trq_meanEff_Engine[Nm]	n_Engine[rpm]	p_InMan[Pa]	p_Rail[bar]	v_Vehicle[km/h]
2	0	1,286351173	753.3748004	1.806724927	736.7779229	22732.842	120.2911209	3.17914E-41
3	0.0010007	1,283528892	753.3704565	1.806617144	736.148984	22716.23925	120.2894093	3.34535E-41
4	0.0020024	1,281254643	753.3660133	1.814807474	735.6940494	22699.65103	120.2870839	3.3119E-41
5	0.0030025	1,279692402	753.3614746	1.823257128	735.4068331	22683.08871	120.28446	3.27878E-41
6	0.0040035	1,278982697	753.3568806	1.832134451	735.279373	22666.55531	120.2816414	3.24599E-41
7	0.0050016	1,279198583	753.3522574	1.841312685	735.3017942	22650.05727	120.2787264	3.21353E-41
8	0.0060028	1,280351633	753.3476302	1.8506137	735.4618861	22633.60302	120.2758075	3.1814E-41
9	0.0069994	1,282398099	753.3430224	1.859851656	735.7449752	22617.20169	120.2729699	3.14958E-41
10	0.0080009	1,285245895	753.3384553	1.868877282	736.1341207	22600.86202	120.2702898	3.11809E-41
11	0.0089999	1,288762142	753.3339477	1.877468676	736.6095761	22584.59151	120.2678335	3.0869E-41
12	0.0100005	1,292782569	753.3295154	1.885437349	737.1497185	22568.39578	120.2656559	3.05604E-41
13	0.011001	1,297118365	753.3251712	1.892647281	737.7305276	22552.27814	120.2637989	3.02548E-41
14	0.0120028	1,30156747	753.320924	1.898801976	738.3262754	22536.23929	120.2622904	2.99522E-41

Figure 3. Generated Dataset

With the aid of ControlDesk, which can be used to configure and control the run-time experiments, the system is executed under the various conditions. Once the real-time execution is completed, the system response to the faults is captured as time-series data and stored in a CSV file containing system variables and data samples. On the other hand, TCs designed in ModelDesk are executed on the target SUT to generate the documented error logs as text data. The TCs should contain the data inputs and the expected outputs. According to the actual system behaviour, the defined expected output is then compared with the measured output. According to the test execution, the results can be either pass or fail. Thanks to the ASM test function of ModelDesk, the TCs are automatically executed and evaluated. Note that the generated report contains key information about the TCs specifications as well as the execution results (passed or failed). Besides, the report contains a failure log describing the occurred failure. Using the logs of the failed TCs, an intelligent model for RCA is developed based on natural language processing and machine learning methods.

5. Result and Discussion

In this section, the results of the data collection for the different FI configurations are presented. Specifically, the generated data is described, highlighting the effects of single and concurrent faults on system behaviour. Besides, the log data of failed TCs generated in the test report are presented.

In the real-time, both the SUT and the controlled system are executed under normal and abnormal conditions according to the aforementioned system and FI configurations. A total of 40 experiments were conducted with single and simultaneous faults in two driving scenarios. In particular, two files as healthy data from ”highway” and ”ftp 75” scenarios, 18 files represent the single fault types and 30 files for simultaneous faults with 15 combinations have been collected. As a result, 50 CSV files have been obtained as HIL testing records.

Focusing on packet loss and stuck-at as an example of fault types, the system behaviour under the injected fault can be captured at the system level on vehicle and engine speed signals. In Figure 4, the stuck-at fault has been injected into the RPM sensor after 10 sec of driving cycle. The engine’s behaviour deviates significantly from 750 rpm in the normal range to 2300 rpm as soon as the fault is activated, which in turn leads to an increase in energy consumption. Until the 180 sec, the control unit is unable to cope with the fault’s occurrence. However, after 180 sec, the system tries to overcome the fault and behave according to the desired behaviour. Then, the deviation and fluctuations take place again from 210

sec and 290 sec, respectively. The reason behind this change is the driving variations and conditions between the acceleration mode, deceleration mode and the steady state mode. This means, in other words, that the control strategy cannot properly mitigate the fault when the vehicle is accelerating and decelerating sharply, as seen in Figure 5.

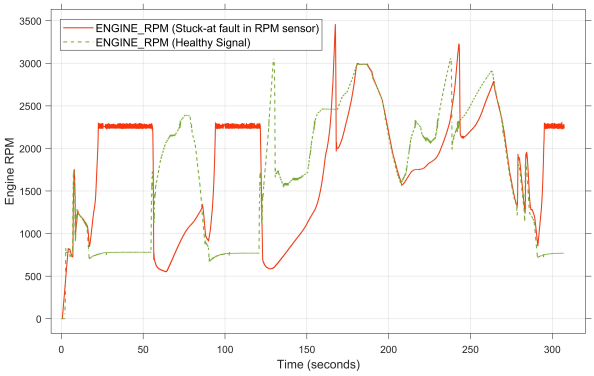


Figure 4. Engine system behaviour under Stuck-at fault

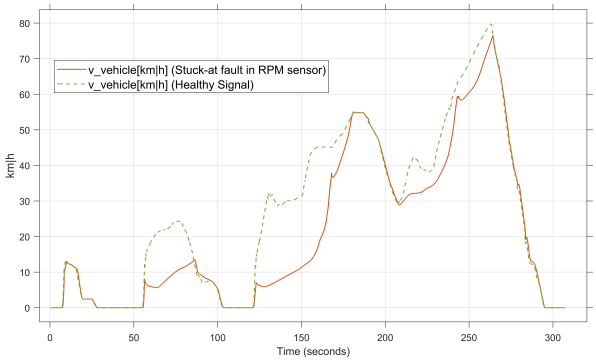


Figure 5. Vehicle system behaviour under Stuck-at fault

On the other hand, by injecting the packet loss into the APP sensor during the driving cycle, the effect of the fault cannot be directly observed on the vehicle speed and the engine speed, as presented in Figure 6. However, from sec 75 onwards, significant fluctuations can be observed. At the system level, the behaviour of the vehicle and the engine deviates from the standard in the form of severe jerking. At this moment, the ECU tries to overcome the signal losses for a certain period of time. That is due to the fact that the behaviour remains in the state of fluctuation while the loss time is less than 3 sec.

According to the scenario "ftp 75", what can be seen in Figure 7 and Figure 8 is the effect of injecting a single fault into the APP and RPM sensors, respectively.

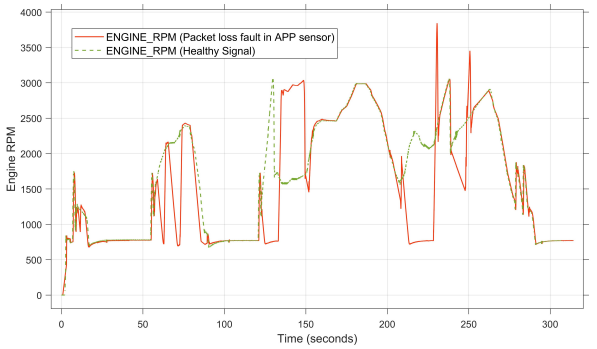


Figure 6. Engine system behaviour under packet loss fault

Specifically, APP sensor signal is manipulated by the stuck-at value, and the effect of the fault is directly observed at the system level as a constant value, i.e., 1160 rpm. On the other hand, the effect of the delay of the RPM sensor signal causes fluctuations of the engine speed between 265-307 sec, as shown in Figure 8. During this period, the ECU cannot perfectly perform the desired behaviour with increasing delay of the received signal, especially when the driving mode is changed.

In the case of the simultaneous faults, two different fault types are injected simultaneously into two different locations. For example, while the stuck-at fault is injected into the APP sensor, the delay fault is simultaneously injected into the engine speed sensor. To put it in another way, two or more factors have contributed to creating a novel pattern in the signals as an effect of the simultaneous FI. Figure 9 illustrates the effect of the simultaneous faults on the vehicle system behaviour, i.e., vehicle speed, between 170-330 sec. The vehicle speed closely follows the desired behaviour up to 170 sec, the time at which the faults are activated. As a result of the large fluctuations shown in Figure 10, the vehicle is unable to accelerate at 190 sec, the time at which the speed drops to 18 km/h, causing a risk of failure. However, the system returns to the acceptable state with minor deviations.

In spite of the fact that the collected data represent comprehensive characteristics of the system behaviour, the generated CSV files contain redundant or useless information. For example, each file contains information about the recording process as well as the data samples. Hence, the collected data should be preprocessed to remove outliers and redundant information. Three steps are applied to the data in the pre-processing phase, i.e., non-usefull information and outlier removal, data scaling and normalisation, and data splitting. After converting the csv files into execl format,

the training variables are selected and the non-useful data is removed. By doing so, the useful data samples are ensured for the development of the ML model. The next step is to apply a normalisation function to all the variables due to the fact that each variable has a different range of values. Thus, all variable values are scaled in the range between [0-1]. The last step is to split the data into three parts, i.e., training data, validation data and test data. Noteworthy, when developing ML models based on supervised learning, the process of labelling the data should take place before splitting the data. During this process, the fault class is assigned to the appropriate data for the classification task. Generally, the data in each part is divided into 80% for training, 10% for validation and 10% for testing part.

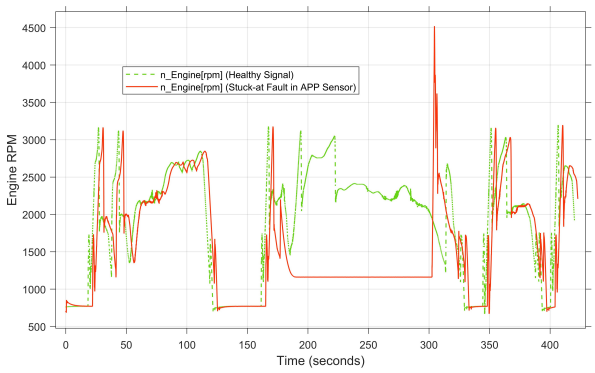


Figure 7. Engine system behaviour under stuck-at fault.

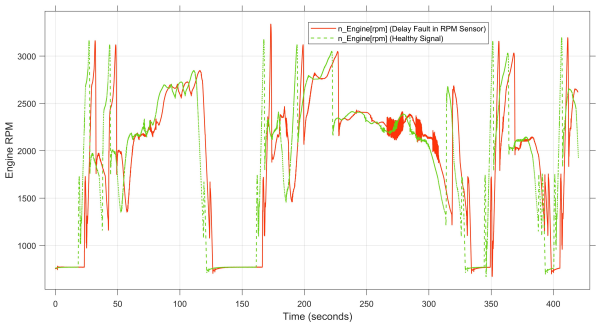


Figure 8. Engine system behaviour under delay fault.

As a result of the TCs execution, where the desired system signals are compared to the observed outputs, a test report is generated. What is of interest in the report is the logs, written in natural language, describing the failures that occurred. One example of the generated test report is shown in Figure 11. As can be seen, a short text is written indicating the failed TCs as information about the causes of the failure. However, the mentioned transcripts are not able to correctly identify the failure

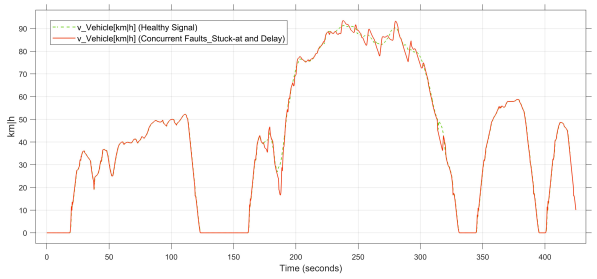


Figure 9. Vehicle system behaviour under stuck-at and delay faults simultaneously

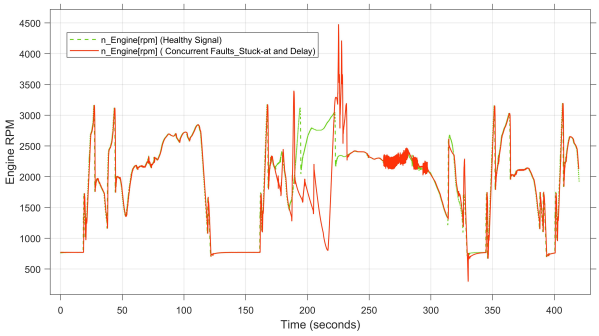
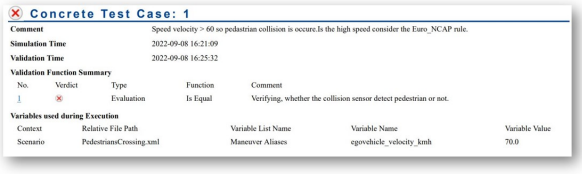


Figure 10. Engine system behaviour under stuck-at and delay faults simultaneously

causes without the expert knowledge of the tester. Therefore, an intelligent model can be trained based on the collected logs so that the RCA process can be performed efficiently without human intervention. Furthermore, the generated logs can be employed to provide additional information to support the decision of the signal-based FDD model. Similar to the time series data, various steps are also conducted on the collected logs as a pre-processing stage before training the model. The key steps in preprocessing text data are tokenization, normalisation, stop word removal and stemming. After applying Natural Language Processing (NLP) techniques to the collected data, it is divided into training three parts: training with 80%, validation with 10% and testing with 10%.

Despite the reliability of the proposed framework in simulating realistic system behaviour in the presence


Concrete Test Case: 1

Comment	Speed velocity > 60 so pedestrian collision is occure.Is the high speed consider the Euro_NCAP rule.
Simulation Time	2022-09-08 16:21:09
Validation Time	2022-09-08 16:25:32

Validation Function Summary

No.	Verdict	Type	Function	Comment
1	✗	Evaluation	Is Equal	Verifying, whether the collision sensor detect pedestrian or not.

Variables used during Execution

Context	Relative File Path	Variable List Name	Variable Name	Variable Value
Scenario	PedestriansCrossing.xml	Maneuver Aliases	egovehicle_velocity_kmh	70.0

Figure 11. Generated report of test case execution

of faults, several complicating factors concerning setup and implementation should be considered. Since three different fault attributes have to be specified in the setup phase, an unlimited number of possible configurations could be created, i.e., an unbounded fault space. For complex software systems, the trade-off between achieving high test coverage and exploring the most effective critical fault is still a challenge. Notably, in this study, the fault test cases were identified based on three factors, i.e., representative and realistic faults to be injected, the impact and rapidity of the injected fault causing failures, and the variety of fault scenarios. Besides the setup effort, a real-time system should be provided to enable real-time execution of the system model along with the framework. For this purpose, in our study, MicroAutoBox II embedded PC (DS1401 Base Board) with 900 MHz processor, 6th Gen.Intel® CoreTM i7-6822EQ, 16 MB memory and 340 ms boot time for 3 MB application are utilized. Based on the debugging run-time information for 10 sec log and intended sampling time of 1 ms, the maximum task execution time is 0.738 ms. Each test case has a test execution time of 40242.6164 ms, including the evaluation period.

Finally, depending on the specification of the log data system, a huge amount of data can be collected from heterogeneous components, which poses another challenge in terms of computational cost of ML/DL model development. Specifically, considering 0.001 sec as the sampling period, several million data samples are generated, including outliers, noise and missing/redundant samples. Therefore, training the ML/DL model not only requires significant effort and time in the pre-processing and training phases, but also demands powerful hardware with high computational capability, such as GPUs or TPUs, to process such a large amount of data.

6. Conclusion and Future Work

A novel framework for generating and collecting representative data for intelligent failure analysis applications during the system validation process is proposed in this paper. To this end, HIL simulation and a fault injection method are used considering real-time constraints. The proposed framework objective is to provide the required dataset for training, testing and validation of DL-based intelligent failure analysis of HIL test records. Innovatively, there is no need for fault modelling within the system architecture in order to acquire the data. Instead, the target faults are injected programmatically in real-time without any modelling effort and time. Moreover, the proposed framework is able to simulate and collect faulty data in different formats, i.e., sequential and textual. As time-series data, the effect of hardware random faults is simulated at the system level in terms of transient and permanent faults. In this manner, the system behaviour under various critical fault conditions can be accurately and reliably captured in real-time. The higher the quality of the captured data, the higher the performance of the developed DL model. In this study, a complex automotive gasoline engine has been selected to validate and demonstrate the proposed approach. The outcomes show the applicability of the approach in representing the effects of single and simultaneous faults at the system level. Simulations of nine different types of sensor and actuator faults were carried out in this study. Besides, combination of two faults at different locations simultaneously was simulated. On the other hand, the textual descriptions of the failed TCs were generated as a result of injecting the faults in real-time during the system executions. To conclude, a high quality dataset with high faults classes coverage can be collected with low effort during real-time simulation.

In the future, the proposed framework will be further improved using automation tools so that the injection process can be performed automatically, effectively and systematically according to the requirements.

References

Abboush, M., Bamal, D., Knieke, C., & Rausch, A. (2022a). Hardware-in-the-loop-based real-time fault injection framework for dynamic behavior analysis of automotive software systems. *Sensors*, *22*(4), 1360.

Abboush, M., Bamal, D., Knieke, C., & Rausch, A. (2022b). Intelligent fault detection and classification based on hybrid deep learning methods for hardware-in-the-loop test of automotive software systems. *Sensors*, *22*(11), 4066.

Badihi, H., Zhang, Y., Jiang, B., Pillay, P., & Rakheja, S. (2022). A comprehensive review on signal-based and model-based condition monitoring of wind turbines: Fault diagnosis and lifetime prognosis. *Proceedings of the IEEE*.

Bafroui, H. H., & Ohadi, A. (2014). Application of wavelet energy and shannon entropy for feature extraction in gearbox fault detection under varying speed conditions. *Neurocomputing*, *133*, 437–445.

Biddle, L., & Fallah, S. (2021). A novel fault detection, identification and prediction approach for

autonomous vehicle controllers using svm. *Automotive Innovation*, *4*, 301–314.

Chen, Y., Chen, S., Zhang, T., Zhang, S., & Zheng, N. (2018). Autonomous vehicle testing and validation platform: Integrated simulation system with hardware in the loop. *2018 IEEE Intelligent Vehicles Symposium (IV)*, 949–956.

dSPACE. (2023). *Experiment and Visualization Software*. Retrieved March 30, 2023, from https : / / www . dspace . com / en / inc / home / products/products.cfm

Fernandes, M., Corchado, J. M., & Marreiros, G. (2022). Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: A systematic literature review. *Applied Intelligence*, *52*(12), 14246–14280.

Garousi, V., Felderer, M., Karapıçak, Ç. M., & Yılmaz, U. (2018). Testing embedded software: A survey of the literature. *Information and Software Technology*, *104*, 14–45.

Gonzalez-Jimenez, D., Del-Olmo, J., Poza, J., Garramiola, F., & Madina, P. (2021). Data-driven fault diagnosis for electric drives: A review. *Sensors*, *21*(12), 4024.

Himmler, A., Lamberg, K., & Beine, M. (2012). *Hardware-in-the-loop testing in the context of iso 26262* (tech. rep.). SAE Technical Paper.

Jordan, C. V., Hauer, F., Foth, P., & Pretschner, A. (2020). Time-series-based clustering for failure analysis in hardware-in-the-loop setups: An automotive case study. *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 67–72.

Jung, D. (2020). Data-driven open-set fault classification of residual data using bayesian filtering. *IEEE Transactions on Control Systems Technology*, *28*(5), 2045–2052.

Kukkala, V. K., Tunnell, J., Pasricha, S., & Bradley, T. (2018). Advanced driver-assistance systems: A path toward autonomous vehicles. *IEEE Consumer Electronics Magazine*, *7*(5), 18–25.

Mihalič, F., Truntič, M., & Hren, A. (2022). Hardware-in-the-loop simulations: A historical overview of engineering challenges. *Electronics*, *11*(15), 2462.

Nair, V. V., & Koustubh, B. P. (2017). Data analysis techniques for fault detection in hybrid/electric vehicles. *2017 IEEE Transportation Electrification Conference (ITEC-India)*, 1–5.

Neupane, D., & Seok, J. (2020). Bearing fault detection and diagnosis using case western reserve university dataset with deep learning approaches: A review. *IEEE Access*, *8*, 93155–93178.

Rengasamy, D., Jafari, M., Rothwell, B., Chen, X., & Figueredo, G. P. (2020). Deep learning with dynamically weighted loss function for sensor-based prognostics and health management. *Sensors*, *20*(3), 723.

Safavi, S., Safavi, M. A., Hamid, H., & Fallah, S. (2021). Multi-sensor fault detection, identification, isolation and health forecasting for autonomous vehicles. *Sensors*, *21*(7), 2547.

Shafiq, S., Mashkoor, A., Mayr-Dorn, C., & Egyed, A. (2021). A literature review of using machine learning in software development life cycle stages. *IEEE Access*, *9*, 140896–140920.

Shokry, H., & Hinchey, M. (2009). Model-based verification of embedded software. *Computer*, *42*(4), 53–59.

Tagawa, T., Tadokoro, Y., & Yairi, T. (2015). Structured denoising autoencoder for fault detection and analysis. *Asian conference on machine learning*, 96–111.

Theissler, A. (2017). Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowledge-Based Systems*, *123*, 163–173.

Theissler, A., Pérez-Velázquez, J., Kettelgerdes, M., & Elger, G. (2021). Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliability engineering & system safety*, *215*, 107864.

Vermeulen, B. (2008). Functional debug techniques for embedded systems. *IEEE Design & Test of Computers*, *25*(3), 208–215.

Yin, Z., Hu, N., Chen, J., Yang, Y., & Shen, G. (2022). A review of fault diagnosis, prognosis and health management for aircraft electromechanical actuators. *IET Electric Power Applications*, *16*(11), 1249–1272.

Zhang, S., Zhang, S., Wang, B., & Habetler, T. G. (2020). Deep learning algorithms for bearing fault diagnostics—a comprehensive review. *IEEE Access*, *8*, 29857–29881.

Zhang, T., Chen, J., Li, F., Zhang, K., Lv, H., He, S., & Xu, E. (2022). Intelligent fault diagnosis of machines with small & imbalanced data: A state-of-the-art review and possible extensions. *ISA transactions*, *119*, 152–171.