FIDGET: Deep Learning-Based Fault Injection Framework for Safety Analysis and Intelligent Generation of Labeled Training Data

Conference Paper · September 2022

DOI: 10.1109/ETFA52439.2022.9921507

4 authors:

Tagir Fabarisov
Universität Stuttgart
14 PUBLICATIONS **64** CITATIONS
SEE PROFILE

Andrey Morozov
Universität Stuttgart
96 PUBLICATIONS **576** CITATIONS
SEE PROFILE

Ilshat Mamaev
Proximity Robotics & Automation GmbH
20 PUBLICATIONS **133** CITATIONS
SEE PROFILE

Philipp Grimmeisen
Universität Stuttgart
10 PUBLICATIONS **11** CITATIONS
SEE PROFILE

FIDGET: Deep Learning-Based Fault Injection Framework for Safety Analysis and Intelligent Generation of Labeled Training Data

1st Tagir Fabarisov
University of Stuttgart
Stuttgart, Germany
tagir.fabarisov@ias.uni-stuttgart.de

2nd Andrey Morozov
University of Stuttgart
Stuttgart, Germany
andrey.morozov@ias.uni-stuttgart.de

3rd Ilshat Mamaev
Karlsruhe Institute of Technology
Karlsruhe, Germany
ilshat.mamaev@kit.de

4th Philipp Grimmeisen
University of Stuttgart
Stuttgart, Germany
philipp.grimmeisen@ias.uni-stuttgart.de

***Abstract*—Since the introduction of the term Cyber-Physical Systems (CPS) in 2006, they came to a long way. CPS are now autonomous and networked systems of systems with state-space exceeding the capabilities of conventional risk analysis methods. Model-based fault injection methods allow assessment of a system's fault tolerance not only during its design phase but also in the course of operation. This allows the evaluation of updates and new modules before deploying such changes to a real system. Such operational model-based fault injection on a system's digital twin can ensure continuous safety throughout all system life cycles.**

Modern risk analysis tools and Machine Learning-based safety methods require vast amounts of representative input and training data. Such methods not only will require mountains of erroneous time-series data from a myriad of operational cycles, but also corresponding fault parameter labels. As the state space of the system component explodes in complexity, it becomes problematic to cover all possible component fault combinations. As such, only those faults that could lead to potential failures or increased risk scenarios are of interest for automated safety assessment methodologies. It is clear that an intelligent and effective model-based fault injection method is required for the operational safety assessment of industrial CPS.

Recently we introduced a new model-based fault injection method implemented as a highly customizable Simulink block called *FIBlock*. It supports the model-based injection of typical faults of CPS components such as sensors, software, computing, and network hardware. In this paper, we proposed a Deep Learning-based approach for model-based fault injection called *FIDGET*. It extends the *FIBlock* with Deep Reinforcement Learning capabilities. We employed a Deep Deterministic Policy Gradient algorithm with Long Short-Term Memory (LSTM) architecture to train the Reinforcement Learning agent to perform the automated search of fault parameters that yield the biggest system response. It allows automatic generation of labeled training data for further use in risk analysis tools or to train fault classifiers. The generated training data consists of errors that lead to the biggest response (i.e., disturbance) of the system.

***Index Terms*—cyber-physical system, error mitigation, fault injection, model-based method, deep learning, reinforcement learning, LSTM, exoskeleton system**

I. Introduction

With the increasing behavioral and structural complexity of modern CPS, the methods for Risk Assessment tasks are becoming more data hungry. For instance, the risk and dependability of wireless sensor networks during its operation can be assessed solely by employing the Big Data techniques [1]. The safety of re-configurable smart factories with Human-in-the-Loop component can be validated by Neural Networks [2]. The dependability of neural networks themselves can be evaluated with injection of possible optical errors to input layer and bit-flips in its hidden layers [3]. All these methods require vast amount of labeled data for training. Recently, we introduced a model-based Fault Injection tool called *FIBlock* [4]. This tool allows injection of various customizable faults into MATLAB Simulink models. Its capabilities were proved on the CPS case study exoskeleton system. Using FIBlocks, we performed Fault Injection experiments with raging fault duration [5]. However, such approach, should it be employed on more complex system of systems, will show its main drawback. The possible combination of various component fault parameters in such system will be too big for brute force iteration or even Monte Carlo simulations. Its complexity would also prevent to perform numerous Monte Carlo fault injection simulations, as it requires great computational power. As such, an automatic, smart, and effective methodology for performing the fault injection experiments is needed.

Contribution: In this paper, we are introducing the Deep Learning-based Fault Injection methodology called FIDGET. It consists of previously presented fault injection tool FIBlock coupled with Deep Reinforcement Learning framework. The environment for Reinforcement Learning agent is a given Simulink model equipped with FIBlocks. The fault parameters are being used as the possible actions (state space) for the Reinforcement Learning (RL) agent. The architecture of RL agent is Deterministic Policy Gradient algorithm with LSTM

layers. The reward is the system response to the injected fault in terms of deviation of a controller signal from the error-free simulation. We applied the FIDGET framework to the exoskeleton case study model and conducted RL-based fault injection experiments. Experiments proved the feasibility of the presented framework. Limitations and useful for further research observations are presented in this paper.

II. Preliminaries

A. Challenges of safety analysis and assurance for Industrial CPS

Industrial CPS are distributed, networked, and intelligent mechatronic systems with sophisticated control software and complex communication protocols. To assess a given CPS safety and evaluate its dependability, it is required to take into account possible critical failure scenarios and parameters of faults that could lead to such an outcome. Model-based Fault Injection (FI) is a methodology for evaluation of a system fault tolerance. Model-based FI allows a software engineer to obtain the knowledge on system dependability and reliability already during design and prototype phases of system development. FI experiments can shine a light on how a controller would handle an erroneous input signal, or which network hardware faults are most critical to the overall system robustness [5]. However, high structural complexity of CPS makes application of FI-based safety evaluation a computational expensive task. To handle this drawback, it is possible to optimize FI by enhancing it with Machine Learning guidance. Automated and intelligent FI guided by e.g. Reinforcement Learning can help to find critical failure scenarios faster and with less computational cost.

Heterogeneity of Industrial CPS components leads to the inability to employ the conventional anomaly detection techniques without adding overhead to the target system. Thus, the Deep Learning-based (DL) anomaly detection and mitigation techniques are being put in the spotlight [6]. They encompass good performance and flexibility by learning to represent the data as a nested hierarchy of concepts within layers of the neural network. DL outperforms traditional machine learning as the scale of data increases [6]. That is indeed the case when dealing with the large amounts of the time-series signal data generated by the various CPS components. Supervised anomaly detection is a class of DL-based methods that learn the boundary between different classes on the labeled training data sets. In theory, they outperform the unsupervised ones. These techniques can provide not only whether an error occurred, but also the type of this error. However, the main difficulties are that the multi-class supervised techniques require labeled training sets consisting of different, numerous erroneous data. Since correct data instances are more common, it is a challenging task to obtain the required amount of the accurately labeled training sets for error types [7]. RL-based FI can be a possible solution for this challenge. After the identification of the most critical fault types, it can help to generate the representative erroneous labeled training data-sets.

In addition to the aforementioned challenges, modern control software in production systems is tend to be replaced by black-box Artificial Intelligence (AI) models. Such models possess almost infinite state space. Their safety and robustness are dictated not just by the structure and architecture, but more so by the data set they were trained on. If such AI-based control systems deployed in a safety-critical production system, it would be almost impossible to provide any meaningful safety assurance. One possible solution can be extensive fault injection experiments. Model-based FI of faults to the components will allow obtaining the labeled training data-sets with erroneous signals. This will train distinguishing possible input data errors that may arise during the operation [3]. Thus, it will allow the resilience validation of such control software to erroneous input data. Another critical aspect of modern CPS is the presence of humans and the high involvement of Human-Machine Interactions. Such CPS with Human-in-the-Loop components are subject to increased safety concerns. They are usually coupled with AI-based software to govern the interaction, e.g. having an overhead camera to recognize human operations and control the corresponding movements of a robot manipulator. An optical error in such a system will not just lead to worse robustness and decreased production, but also to a possible human operator injury.

B. Proposed solution

To solve the aforementioned challenges of Industrial CPS, we have developed the FIDGET framework. The main goal of FIDGET is to automatically generate labeled training data for risk assessment methods and tools, as well as for Neural Networks. Because of the increased complexity of modern CPS, the state space of systems is becoming quasi-infinite. Each subsystem can consist of numerous heterogeneous components. Every CPS component can have different types of faults common for a given type of component. Some of these faults can trigger further error propagation and cause a system failure. However, many more will be compensated by the control loop or will cause only tolerable deviation from expected system behavior. As we showed in the previous study, sometimes the main concern is not only the fault magnitude, but also how frequent faults are being activated [8]. As such, it is important to explore not just the parameters of a single fault, but combinations of several faults. This will provide valuable insight into the internal specifics of fault activation and error propagation of a given system. The previously presented FIBlock tool allows such chained fault injection. Hence, the FIDGET framework can explore multiple faults as well.

III. FIDGET framework

A. Top-level structure

The presented FIDGET framework is developed in the MATLAB Simulink environment. The top-level structure of the proposed solution is shown in Figure 1. Following are the steps for application of the FIDGET framework:

1) **Initial setup.** The usage of framework starts with the selection of possible points in a given system where the FIDGET can inject faults. It is possible to specify which types of faults should be explored if there is available knowledge about the target system. The user can manually cap or restrict the fault value. For example, if desired fault magnitude shouldn't exceed a certain threshold.
2) **Training.** Reinforcement Learning Agent (RL agent) performs guided FI. The goal of FIDGET here is to find the fault parameters that will yield the maximum system response while not causing system failure, i.e. minimizing the intrusion to the system.
3) **Training data generation.** After maximizing the reward function, FIDGET performs final fault injections with the discovered fault parameters. During this step, FIDGET will not only output erroneous signals, but also labels representing fault types and the exact time step at which the faults were activated. This automatically generated labeled training time-series data can be used for consecutive training of e.g. ML-based fault detectors and classifiers, and for risk assessment tools.

The architecture of FIDGET is shown in Figure 2. It consists of two main parts: Fault Injection Environment and Reinforcement Learning module. Fault Injection Environment is a Simulink MATLAB Function block, that is the core component of the FIBlocks (Section III.B). The Reinforcement Learning part (Section III.C) consist of an RL agent that changes the fault parameters (*Actions*) based on the *Reward Function* and *Observations*. The Root Mean Square Error (RMSE) metric, which is a deviation from the error-free signals and erroneous signals, is being used to calculate the *Reward*. The user-defined input signals are playing the role of *Observations*.

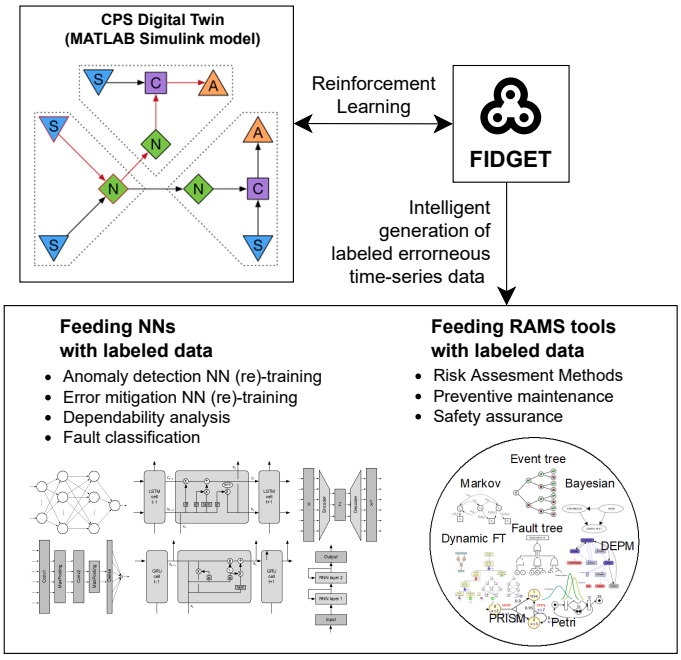

Fig. 1. Top-level structure of proposed solution.

B. Model-based Fault Injection

The FIBlock consists of a custom Simulink MATLAB Function block that processes the incoming signal and injects faults according to the user-specified parameters. FIBlock supports heterogeneous fault types intrinsic to CPS system components such as stuck-at, bias, sensor freeze, bit-flip, package drop, and time delay. For each type of fault it is possible to specify the fault value, e.g. a magnitude of bias. FIBlock allows specification of a fault activation method: at exactly chosen time step, with a probability of occurrence, or with *mean time to failure*. Additionally, the fault duration can be specified in terms of the exact number of time-steps or using *mean time to repair*. More details are available in the introductory paper [5].

As it is discussed before, the injection of faults in multiple locations is called multiple-point fault injection. The situation when the activation of one fault triggers the activation of another fault via error propagation is called chained faults. FIBlock supports both multiple-point and chained faults. As such, it is possible for the FIDGET to find not only fault parameters that yield the most system response, but also the best placement of said faults.

The so-called *Fault space* is a representation of fault parameters as coordinates in a multi-dimensional space. For the ease of results validation in this research, we are limiting the fault space to two fault types and fixed faults placement.

C. Reinforcement Learning

The architecture of the FIDGET framework is shown in Figure 2. The main feature of FIDGET is it's high-level of abstraction and versatility. It can explore the fault space and search the fault parameters regardless of the system type and signal nature. The possible fault parameters are restricted only by the capabilities of the FIBlock. The search is dictated solely by the incoming signal readings and the fault parameters themselves. Hence, the reward function is calculated based on *deviations* from error-free signal and *intrusion* of fault injection. The intention is that the found faults will yield maximum system response while being as short and as low-magnitude as possible. Following State of the art [9], we have employed the Deep Deterministic Policy Gradient algorithm with Recurrent Neural Network (RNN). For the architecture of RNN we have chosen LSTM. The learning and architecture parameters are shown in Table 1. The RL agent has two LSTM models, one for *critic* and one for *actor*. *Critic* takes as input two concatenated signals: *observations* and *actions*. *Actor* takes as input the *observations* and returns *action* signal for FIBlock. The size of output layer is equal to the number of actions, i.e. fault parameters.

TABLE I
ARCHITECTURE AND TRAINING PARAMETERS OF THE DEVELOPED DEEP REINFORCEMENT LEARNING AGENT.

Parameter	Value
Number of sequence input layers	2
Number of concatenation layers	1
Number of hidden LSTM layers	2
Number of units per hidden layer	8
Size of output layer	4
Activation	Hyperbolic tangent
Training batch size	32
Number of epochs	200

Environment: The environment of FIDGET is a given MATLAB Simulink model equipped with FIBlocks. More about that in Section III.b. As it was discussed before, it is required to specify the placement of FIBlocks. The placements are possible points of fault activation, e.g. sensor output signals. Since the Reward function is calculated based on the fault parameters and signal deviations, it is necessary to provide the Access Points (AP) placement in the system. That is, from which signals to feed the RL agent with observations from the environment. The choice of AP placement should be dictated by the goal of data generation. For example, in our experiments, we were evaluating sensor faults and the corresponding controller response. A snippet of the case study exoskeleton model equipped with FIDGET is shown in Figure 3.

Actions: The actions of RL agent control the parameters of FIBlocks. RL agent can turn on and off particular FIBlock. This enables the exploration of different placement of faults. Each of the possible fault injection placements has its fault space. The axis of fault space are fault parameters:

- ***Fault type.*** What kind of fault will be injected. Stuck-at-0; Bias; Sensor freeze; Bit-Flip; Package drop; Time delay.
- ***Fault value.*** Each type of fault could take different values depending on its nature. E.g., Bias can be either positive or negative natural number.
- ***Fault event.*** When a fault will be injected. With probability (stochastically). Mean Time to Failure (using statistic metric). Deterministic (on exact specified time-step).
- ***Fault effect.*** How long a fault will last. Once (fault will last only one time-step). Constant time (specified number of time-steps). Infinitely (once the fault activated, it will be injected until the end of simulation). Mean Time to Repair (using statistical metric).

Reward function: As it was mentioned before, the FIDGET goal is to find the fault parameters that will yield the most system response. But that is not as trivial as it may seem. For example, the RL agent can just change the fault parameter to Bias of magnitude 100. Of course, such a fault will for certain lead to a system failure. But that would be **a)** not realistic, as such fault is not possible in real systems; **b)** it will prevent the system operation. Such fault won't generate any useful training

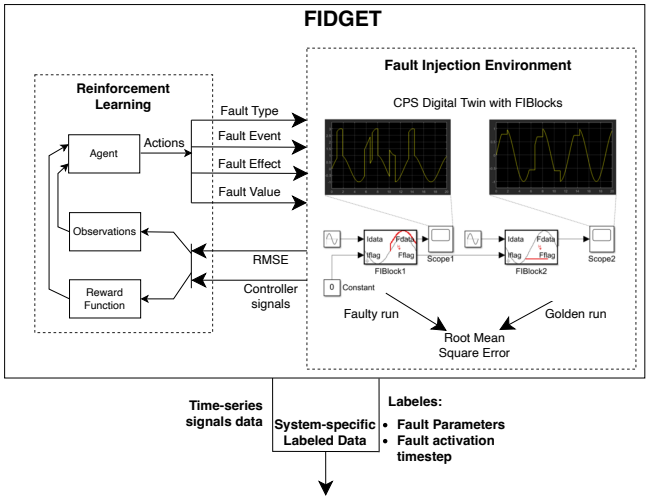

Fig. 2. The architecture of FIDGET framework.

data. On the other hand, having faults (albeit with noticeable values) that are not leading to any system state deviations is as useless. For the reward function, it is necessary to formalize both system response and system intrusion. This is achieved with the following formula: $R_t = RMSE_t * 0.2 - Fb_t * 0.8$, where *RMSE* is Root Mean Square Error between error-free signal and potentially erroneous signal during fault injection at a given time-step; *Fb* is fault effect - that is fault magnitude (More in Section III.b). That way, the RL agent will maximize the deviations of system signals while simultaneously will minimize the intrusion to the system, i.e. fault effect. The resulting faults will yield the most system deviations while being less in duration and magnitude. This will reflect the intrinsic nature of heterogeneous CPS components that can have short transient signal errors.

The action signal is defined as a tulip:

$$
\begin{aligned}
Action &:= (F, T, R): \\
F &:= \{f_1, \ldots, f_n\}, \\
T &:= \{t_1, \ldots, t_n\}, \\
R &\subseteq (F \times prob \times T \times enable)
\end{aligned}
$$

F	is a set of links to fault injectors in the system;
T	is a set of possible fault types;
R	is a relation that maps probability and fault types to fault injectors;
$prob$	is the probability of failure: prob $\in [0; 1]$;
$enable$	is the boolean flag of a fault activation.

IV. EXPERIMENTS

A. Case Study model

To validate the capabilities of FIDGET, we applied it to an exoskeleton case study system model. We employed a safety-critical SafeLegs exoskeleton MATLAB Simulink model that was previously used for the demonstration of our Deep

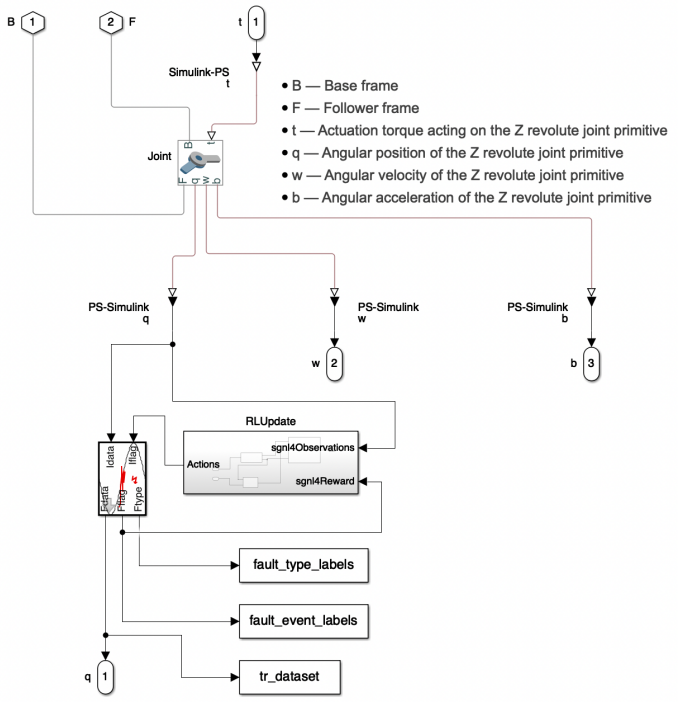

Fig. 3. The right knee subsystem of the case study Simulink model equipped with FIDGET.

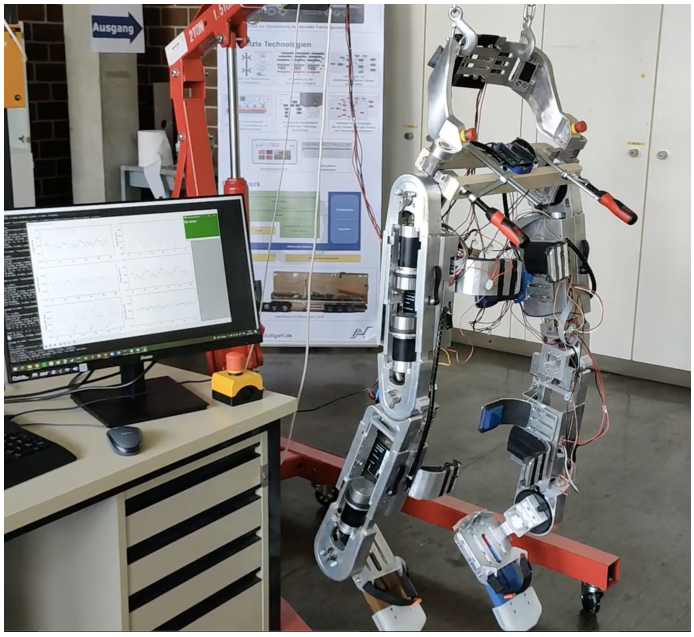

Fig. 4. SafeLegs prototype.

Learning-based error mitigation framework [8]. The SafeLegs is a lower-limb exoskeleton demonstrator system equipped with safety features. It is the result of collaboration between Uni Stuttgart and KIT. This system is based on the SafeLegs prototype developed within the EXO-LEGS EU-AAL-project. It is a supportive lower-limb 6-DOF exoskeleton system. A photo of the SafeLegs prototype is shown in Figure 4 It consists of six actuators - brushless motors in the hips, knees, and ankles. Sensor signals are providing feedback from the motor encoders to the main system controller via the CAN bus and CANopen protocol. This system is a perfect example of complex CPS with components that can have various heterogeneous faults. Possible component faults are ranging from sensor freeze network delay to controller memory bit-flips.

Previously, via extensive fault injection experiments, we evaluated fault compensation capabilities of the SafeLegs [5]. We discovered the spatial and temporal tolerable thresholds for different sensor faults. The injected faults lead to 'Sensor freeze' and 'Stuck-at 0' erroneous signals. Next, we developed a DL-based failure prevention system [8]. We evaluated it in online manner on the SafeLegs case study Simulink model. In these studies, the fault injection was performed using FIBlocks automated with scripts.

As discussed in Section II, the generation of labeled training data sets with representative and realistic faults is a pressing problem that is hard to tackle. In the aforementioned research, we generated the training data almost manually, using a script that was manipulating only fault duration. Using domain knowledge, we knew exactly which faults to inject and where. However, even in this case, the fault injection experiments took a good part of the research. Moreover, we can't know if the explored faults were the most critical for the given system. Thus, we have applied FIDGET to the SafeLegs Simulink model to discover the most critical fault parameters and to generate the representative training data-sets.

B. Experimental setup

The SafeLegs Simulink model equipped with FIDGET is shown in Figure 3. Observation input is taken from control signals. The reward is calculated using fault parameters output and observed input signals. Based on the calculated reward, the RL agent outputs the actions that are being used as trigger input for the FIBlock.

To keep the experiments in line with our previous findings, we limited the placement of possible faults to the right knee joint angular position sensor signal. The possible types of faults were chosen between Stuck-at-0 and Sensor freeze. The stop criteria were chosen based on the situation when the controller couldn't compensate the injected fault, which lead to system failure. After each episode, the RL agent is changing fault parameters based on the calculated reward function. Thus, it explores the fault space more efficiently, as the maximization of the reward function leads to fault parameters that yield the most system response. After the training stops, we extracted the fault parameters from the episodes with the maximum *Reward*. With these fault parameters we conducted a final set of Fault Injections and obtained labeled training data that contains the most critical errors. This time-series data will be used for further research in training of DL-based fault classifiers for CPS.

C. Results and discussion

Using Reinforcement Learning for fault space exploration allowed us to discover safety critical faults and additional important knowledge about the safety aspect of the case study system. A snippet of obtained datasets with discovered faults is shown in Figure 5. Previously, we discovered that the timing of fault activation is critical for the success of fault mitigation. But only in the sense that faults are most critical if they are activated too frequently. Upon the evaluation of the new results, we were able to further clarify and refine that. We have found out that the severity of system response (value of reward function) is correlating with the exoskeleton's movement gait cycle. For example, when it's a peak of angular position signal, the Stuck-at fault will lead to the most critical situation, while the Stuck-at-0 fault will not. That is because immediately after this peak, the left leg starts its movement, while the right knee returns to the 0-degree position. Another example is the sensor freeze fault. Such fault wouldn't lead to any critical situation at all during this part of the gait cycle. That is, as long as it's no longer than 0.83 seconds. This is longer than the threshold that was discovered in our previous research using scripted fault injections.

Conducted experiments proved the feasibility of the presented FIDGET framework. Because the labels for the generated data sets are provided by the fault injectors themselves, they can be used for the subsequent risk evaluation and training of Neural Networks. Accompanied by domain knowledge, this methodology can be applied to almost every Simulink model. However big and complex the given system is, FIDGET only requires a control signal for observations and corresponding erroneous signals to calculate the system response. Other variables for the calculation of Reward are based solely on the FIBlocks output signals, which are tightly integrated into the FIDGET.

However, this is just one side of the coin. As with every black-box approach, e.g. DL-based fault mitigator, the performance of that is as good as the level of expertise of the user. The changes in fault parameters are being controlled solely by the observations. If the input signal for the FIDGET is chosen poorly, the reward function won't reflect the system deviations appropriately. This will lead to missed critical fault parameters.

V. CONCLUSION

In this paper, we presented the FIDGET framework. It couples the flexibility of model-based fault injection with the intelligence and autonomy of Reinforcement Learning. We have proved its capabilities to generate labeled training datasets with representative errors. It was also proven to be a powerful framework to automate the safety evaluation of complex Simulink CPS models.

We employed FIDGET to the complex exoskeleton case study system. Conducted series of the experiments broaden our knowledge about the safety and dependability aspects of this system. This will allow us for further research in the field of Deep Learning-based fault mitigation for Cyber-Physical Systems with Human-in-the-Loop.

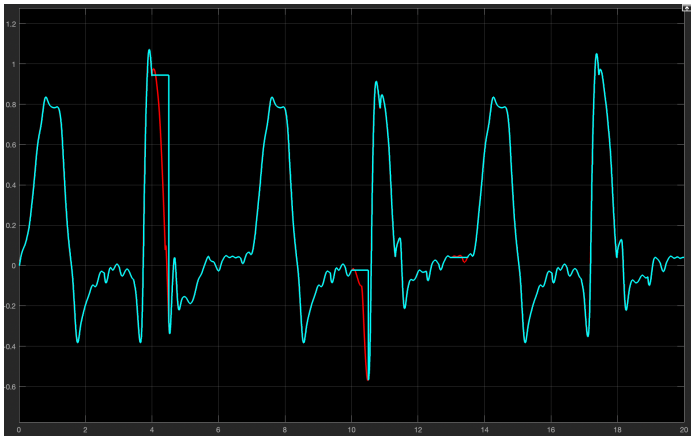

Fig. 5. A snippet of obtained sensor signals. Blue - erroneous signals. Red - corresponding signals after controller compensation efforts.

The next step will be to compare the performance of different architectures of Neural Network trained with data acquired using FIDGET and with traditional approaches.

VI. ACKNOWLEDGEMENTS

This research is supported by the Bundesanstalt für Arbeitsschutz und Arbeitsmedizin (BAuA, Germany) funds, project F 2497.

REFERENCES

[1] Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The computational limits of deep learning. arXiv preprint arXiv:2007.05558.
[2] Sóti, G., Mamaev, I., & Hein, B. (2020, October). A Modular Deep Learning Architecture for Anomaly Detection in HRI. In International Conference on Interactive Collaborative Robotics (pp. 295-307). Springer, Cham.
[3] Morozov, A., Valiev, E., Beyer, M., Ding, K., Gauerhof, L., & Schorn, C. (2020). Bayesian Model for Trustworthiness Analysis of Deep Learning Classifiers. In AISafety@ IJCAI.
[4] Fabarisov, T. (2020) Fault injection block. https://github.com/Flatag/FIBlock/.
[5] Fabarisov, T., Mamaev, I., Morozov, A., & Janschek, K. (2021). Model-based fault injection experiments for the safety analysis of exoskeleton system. arXiv preprint arXiv:2101.01283.
[6] Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407.
[7] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58.
[8] Fabarisov, T., Morozov, A., Mamaev, I., & Janschek, K. (2021, November). Deep Learning-Based Error Mitigation for Assistive Exoskeleton With Computational-Resource-Limited Platform and Edge Tensor Processing Unit. In ASME International Mechanical Engineering Congress and Exposition. American Society of Mechanical Engineers.
[9] Moradi, M., Oakes, B. J., Saraoglu, M., Morozov, A., Janschek, K., & Denil, J. (2020, June). Exploring fault parameter space using reinforcement learning-based fault injection. In 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W) (pp. 102-109). IEEE.