FliPSi: Generating Data for the Training of Machine Learning Algorithms for CPPS

Maria Krantz[1], Niklas Widulle[1], Jonas Ehrhardt[1], Anna Nordhausen[1], Artur Liebert[1], and Oliver Niggemann[1]

[1] *Helmut-Schmidt University/University of the Armed Forces, Hamburg, 22047, Germany*
firstname.lastname@hsu-hh.de

ABSTRACT

Anomaly Detection, the recognition of faulty behavior, and diagnosis, the process of identification of the root cause of a fault, in Cyber-Physical Production Systems (CPPS) are complex tasks, due to the increasing complexity and modularity of modern production systems. But the increasing amount of data, generated by sensors, offers a solution: Machine Learning (ML) can be used to automatically generate models for Anomaly Detection and diagnosis based on data — at least in addition to manual models. While nowadays huge data sets for CPPS exist, they mainly cover repetitive OK situations and lack fault modes, which is problematic for the training of ML algorithms. In this work we present a possibility to overcome this problem by generating data with a simulation of modular CPPS. For simulation we use the game engine Unity, which has already been employed to generate data for the training of self-driving cars and the simulation of robots. Within Unity, modules of CPPS are designed and programmed. Afterwards they are combined into complex CPPS, which are then used for simulation using the built-in physics engine of Unity. The built-in physics engine is not constrained to certain applications, but can adequately simulate a wide variety of scenarios, allowing for the combination of different simulation types. The generated data can be exported for training purposes. In the simulation, faults can be inserted and data sets containing faults and normal behavior can be exported. This way, ML algorithms can be trained for fault detection and diagnosis. The data is especially useful for pre-training of ML algorithms, which can later be fine-tuned using data sets from the real CPPS. This way, the problem of repetitive data sets which lack data for fault cases can be overcome. Furthermore, the simulation can be utilized as a test environment for newly developed ML algorithm before they are employed in real systems.

1. INTRODUCTION

Modern production facilities are complex systems of interconnected machines controlled by computational algorithms. These systems are referred to as Cyber-Physical Production Systems (CPPS) (Monostori, 2014). Due to the increasing complexity within these systems, tasks such as Anomaly Detection, maintenance decisions and diagnosis in case of fault need to be automated and are often carried out with the help of Artificial Intelligence (AI). Improvements in data acquisition and computational power have also led to the increased use of Machine Learning (ML) algorithms.

Anomaly Detection refers to the process of identifying behavior within a system that does not conform with a predefined, expected behavior (Chandola, Banerjee, & Kumar, 2009). With the introduction of ML algorithms in this field, Anomaly Detection refers more to the identification of data points which deviate from the bulk of data (Pang, Shen, Cao, & Hengel, 2021). To train these algorithms, large data sets are necessary to define what the normal behavior of a system looks like. Supervised, unsupervised and deep-learning strategies have been employed to achieve this goal (Angelopoulos et al., 2019).

Once an anomaly is detected, its cause needs to be located. This process is termed diagnosis. It aims at identifying the root cause of a fault in order to enable repairs and return the facility to normal operation (Gertler, 2017). Often, diagnosis is performed with the help of AI to identify the root cause using model-based diagnosis, which needs a human expert to formulate the model. However, there have been approaches to use ML methods for diagnosis, for example by combining model-based diagnosis with machine learning (Bunte, Stein, & Niggemann, 2019).

All ML approaches need large data sets covering a variety of OK and fault modes for training, but many available real data sets do not cover fault scenarios, since they do not appear frequently in real CPPS. A possibility to overcome this problem is to use simulated data sets. However, this requires a realistic simulation of the CPPS to be analyzed. Many simulation

tools currently employed in the simulation of CPPS do not adequately represent all aspects of a CPPS and are therefore not capable of simulating realistic data sets.

Here, tools from the game and media design industry can be employed to adequately simulate a given CPPS and generate realistic training data sets based on the simulation. Specifically, game engines are useful for simulation, since they provide a built-in physics engine to simulate interactions between components and, through embedded programming languages also allow the adaptation of components to the needs of the simulation problem.

In this work, we present FliPSi, the Flexible Production Simulation, based on the game engine Unity. This paper introduces a simulation tool, which is currently under development in our group. We show some preliminary simulation views and describe the tool. FliPSi is a simulation environment for modular CPPS, which can be used to generate realistic training data sets for (pre-)training and testing of ML algorithms.

With FliPSi, models of CPPS can be designed and simulated. Models can be designed for many different cases, such as rarely occurring faults. Even if all fault modes could be anticipated before and, therefore, explicitly modeled, it would still be necessary to recognize and categorize these faults once they occur. For this task ML algorithm employed, as it is not possible to pre-programm every possible fault signature.

When data for a certain ML task is limited, using data sets from similar tasks can be helpful as a pre-training exercise. This is the aim of transfer learning (Torrey & Shavlik, 2010), which is often employed in e.g. image categorization (Shaha & Pawar, 2018) or language processing (Devlin, Chang, Lee, & Toutanova, 2018). An algorithm is first presented with one kind of task, e.g. telling apart photos of cats and dogs, and in a second step has to solve a different, but similar task like classifying photos of other animals. It is assumed that many factors which are necessary to fulfil the first task will also be helpful in the second, i.e. in the earlier example both methods correspond to each other in the layers for edge detection. Since often not enough real data of CPPS is available, using simulated data es a pre-training exercise could help alleviating blocking issues in using ML methods for diagnosis of CPPS.

Another important aspect of developing ML algorithms for anomaly detection and diagnosis in CPPS is the testing of newly developed methods. These cannot be directly tested on an industrial CPPS. Here, using a simulation model of the real system can be useful.

FliPSi has an intuitive user interface and, since it is based on Unity, is easily modifiable and extendable. This makes it useful for education purposes. It can be used to teach aspects of CPPS as well as subjects such as diagnosis or machine learning.

With these aspects in mind, we pose the following Research Questions (RQ):

RQ1: How are ML algorithms currently employed in the field of anomaly detection and diagnosis?

RQ2: Which limitations are present in currently existing real data sets and data generation approaches for the training and testing of ML algorithms for anomaly detection and diagnosis in CPPS?

RQ3: How can these limitations be addressed by employing FliPSi?

With FliPSi we present a possibility to generate realistic training data sets which we think will help overcome a blocking issue in the current use of ML algorithms for diagnosis of CPPS. FliPSi is a modular simulation of CPPS, which combines functional and design modeling and allows the generation of data sets featuring different types of faults. Thereby, FliPSi can be employed to generate realistic, variable data sets of CPPS and provides a testing environment for ML algorithms.

The rest of the paper is structured as follows: Section 2 will briefly discuss the State of the Art of the use of ML in Anomaly Detection and Diagnosis, as well as deliver a short overview of the current use of game engine within engineering. Next, Section 3 discusses the limitations of currently available data sets for training of ML algorithms in anomaly detection and diagnosis, while Section 4 will present solutions to these problems using FliPSi. Finally, Section 6 will summarize the findings and provide an outlook.

2. State Of The Art

2.1. Use of Machine Learning in Anomaly Detection

Due to the complexity of modern CPPS, Anomaly Detection is often carried out with the help of ML algorithms. Different approaches have been suggested, depending on data availability. One main distinction is between Supervised and Unsupervised Learning methods (Hastie, Tibshirani, Friedman, & Friedman, 2009). When no or very limited data for fault cases and anomalies is available, unsupervised learning strategies need to be used. Often, autoencoders (Lindemann, Fesenmayr, Jazdi, & Weyrich, 2019) are employed to classify anomalies in this case. This approach has been further researched, for example by using an nonlinear autoencoder for dimensionality reduction and Anomaly Detection, by exploiting the neural-network autoencoder's dual feature of performing dimensionality reduction, while simultaneously delivering a criterion for decisions on Anomaly Detection, namely its reconstruction error (Eiteneuer, Hranisavljevic, & Niggemann, 2019).

Supervised learning methods can be employed when data sets which contain anomalous behavior are available. These can be methods of deep learning, which are highly suited for data classification and feature extraction (Khan & Yairi, 2018), or methods such as decision trees (Kaparthi & Bumblauskas, 2020) or support vector machines (Pittino, Puggl, Moldaschl, & Hirschl, 2020).

A two-step approach to detecting anomalies in industrial processes, which first classifies the phase of the process and then whether the data is "Expected", "Warning", or "Critical", has been suggested (Quatrini, Costantino, Di Gravio, & Patriarca, 2020). As a classifier they used the decision forests algorithm and the decision jungle algorithm. The advantage with this method lies in the classification of production phases by the algorithm, which allows better prediction.

The use of structured neural networks, which combine an event ordering relationship based structuring technique with deep neural networks, has been proposed both for supervised and unsupervised learning (Liu et al., 2018). With the structuring process, the weight initialization in the deep neural network can be determined before training, which increases the accuracy of the learned network.

2.2. Use of Machine Learning in Diagnosis

Diagnosis is often based on models, which need to be constructed based on expert knowledge. However, approaches to use ML methods for generating these models exist. In (Bunte et al., 2019) a learned model of a small production system is employed for diagnosis. The algorithm first identifies the modes of the system and subsequently learns the quantitative behavior for each of them, thereby enabling faster diagnosis in a mode-based system, compared to traditional model-based diagnosis.

The combination of physics-derived models with ML methods is referred to as hybrid modeling. It has received growing attention in the past few years to address the challenges of modeling CPS. Different methods of integrating physics equations into ML models have been tested, such as using physics-based equations as a regularization term in the loss function of a neural network or physics-based pre-processing of data (Rai & Sahu, 2020).

ML approaches have also been employed to directly diagnose systems, such as induction motors (Shao, Sun, Wang, Gao, & Yan, 2016). A deep-belief network was employed to learn features from vibration signals to classify failure modes. However, to the best of the authors' knowledge, no such algorithm has been developed for CPPS.

2.3. Data Generation Approaches

ML algorithms can use real data sets collected from CPPS for training purposes. However, often the data needed for training is not available in sufficiently large quantities. Therefore, simulated data from models of CPPS is often used.

Models can be constructed in a variety of modeling languages. One of the most common ones is Modelica, which can be simulated in several simulation environments, such as OpenModelica (Fritzson et al., 2005), an open source simulation environment. Modelica is mostly used to simulate different aspects of the physics within a system, such as electrical, thermal or mechanical parts. Simulation of step-wise processes is not easily possible. Another option for modeling of CPPS is Simulink, a block-based modeling language based on Matlab. It is most often employed for modeling and simulating mechatronical systems and can be used to design and program software for these systems. Simulink is proprietary and models cannot easily be exported to other simulation environments. Similarly, Labview is also a block-based modeling language mostly used for software generation and testing. Furthermore, other simulation environments exist, which are mostly proprietary software tailored to specific needs, such as Plant Simulation by Siemens or ExtendSim, which are often used in logistics simulation (Bangsow, 2020).

Generating data sets for ML training and evaluation is also an ongoing field of research. In the field of Anomaly Detection of cyber attacks, the Electra Dataset was generated, which was used to evaluate cybersecurity in an electric traction substation in the railway. However, this data set was generated using a real electric traction substation and cyber attacks were simulated to generate a useful training data set (Gómez et al., 2019).

The BeRfiPl benchmark is a simulation environment for generating CPPS datasets from the process engineering domain (Ehrhardt et al., 2022). The simulation environment is based in OpenModelica. Generated datasets include hybrid systems, various faults, varying system complexity, complex dependencies and recurrences (Ehrhardt et al., 2022)

2.4. Use of Game Engines in Data Generation

Game engines are already employed in modeling of engineering systems, particularly for the training of algorithms for self-driving cars. Especially the capability of game engines to generate visuals is useful to generate training data sets, which can be combined with a simulation of the mechanics of the vehicle's engine using the physics engine (Rong et al., 2020). Another similar approach was used to design CARLA, a simulation environment for urban self driving vehicles. It can simulate realistic scenarios in cities, including building, people and weather conditions (Dosovitskiy, Ros, Codevilla, Lopez, & Koltun, 2017).

Game engines are also already in use as simulation tools in the robotics field, where the Robot Operating System (ROS) is used to develop robots. The game engine Unity provides an

interface for ROS to allow development of robotics software within Unity. This has been employed to develop a simulation environment which can be used to train robots to assemble furniture and is used as a benchmark for deep reinforcement learning algorithms (Lee, Hu, & Lim, 2021).

3. Limitations of Current Approaches

In the following we first cover data sets and then data generation approaches. A data set is by its nature limited in that it is non-interactive, however, it is the only option for data from real CPPS. Data generation uses a simulation to create data sets. This allows not only more flexibility in what the data sets should represent or cover, it also enables direct interaction with the environment. The later can be useful for some approaches based on reinforcement learning, e.g. (Lee et al., 2021).

3.1. Real Data Sets

The main advantage and use case of a real data set is that it represents the real world and can be used as a ground truth. Cherry picking or manipulation aside its clear this advantage holds over all other approaches and means that they need to be used as a final reality check, for verification and validation (Kleijnen, 1995).

However, they have a number of drawbacks. In real systems faults or errors are sparse, therefore in many real data sets they only make up a tiny amount of all data points (Zhang et al., 2022). They also will not cover most possible faults: Faults in production systems can be costly or even dangerous, most fault cases will not be observed by chance and creating a fault for test purposes might also be infeasible.

Additionally, interaction with a real environment is usually impossible, as stated above. These rules out application of approaches that require feedback, like reinforcement learning.

Real data sets are often hard to come by. The data generated by a factory is typically a well guarded trade secret of the operating company. Even if they are used in a research project, the data itself often remains undisclosed. This is especially true in a critical industry like defence. An overview of existing data sets and their fields of application has been compiled by the Fraunhofer IPT (Krauß, Dorißen, Mende, Frye, & Schmitt, 2019).

Real world systems are not easily modified. This makes testing of new approaches difficult. The total amount of data in a real data set could also be insufficient for many ML algorithms. While modern CPPS create a vast amount of data, the time span covered is naturally limited.

3.2. Simulation and Data Generation

Simulations help to alleviate these drawbacks. Here we can categorize them with functional simulation software like Modelica and Simulink on one side and specialized, often commercial tools like Plant Simulation on the other.

As introduced in Section 2.3 tools like Modelica and Simulink are typically used to create functional models of systems. The created models can cover fault cases and can be used for Anomaly Detection or even diagnosis. The models can only depict what was explicitly added to them. This is relevant as in reality, a lot of faults are the result of previously unknown behaviour or interaction. Errors only occurring by chance are not covered and even random or stochastic behaviour has to be explicitly modeled. When used for data generation, the resulting data sets are as a result unnaturally clean. This is a real issue for ML which relies on proper variance of the input data.

Functional models are very capable to depict complex relations between different values. Again these have to be explicitly designed, if there are unknown causal relations in the actual system they will not be present in the simulation. This makes diagnosis of them impossible. Real errors might be the result of very different systems interacting with each other, a purely functional simulation might not cover the actual cause. For example, a product can get stuck on conveyor belt because of its shape. This is almost impossible to model without the use of a 3D simulation. Such a simulation can however be added to Simulink through the use of Simscape (Miller & Wendlandt, 2010).

An alternative are specialized, usually commercial, tools. The software introduced in Section 2.3, Plant Simulation, originates from a discrete event simulation but is extended with a fully 3D visualisation. While the actual feature set of these products obviously varies, there are a few common drawbacks. The software is closed source, which might raise questions about the accuracy or viability of the simulation results. The software has only limited options to modify it: CPPS are very diverse and a fixed software might not come equipped with the tools to cover each specific use case. Other issues might arise with using the software to create data sets. Depending on the tool the options to interact with the software for closed loop control might be limited, and even exporting data sets could have limitations.

As mentioned above, challenges arise when merging ML and simulation applications. This happens due to the fact that there are different terminologies used in the applications. On the one hand, there are many experts who deal with the creation of simulations and have thus increased their understanding of processes. On the other hand, there are experts who deal with the integration of ML applications and thus focus on data processing. (von Rueden, Mayer, Sifa, Bauckhage, &

Garcke, 2020)
Unexpected behavior (e.g., a slightly moved component) is difficult to implement variably, realistically and without explicit programming in simulations. Since ML is very much based on data diversity (e.g., oscillations, noise, etc.), the challenges listed previously arise here, among others. Likewise, this heterogeneous reality leads to difficulties in adapting ML models for simulations, due to the different focus of process and data diversity.

4. SOLUTIONS PROVIDED BY FLIPSI

To address the problems formulated in the preceding chapter, we developed FliPSi, a simulation environment for CPPS. FliPSi is implemented within the game engine Unity which provides possibilities of modeling and simulation that bypass problems, especially in functional modeling of CPPS and in the combination of different types of faults.

4.1. How FliPSi is Designed

FliPSi can be executed within the Unity development environment or as an application similar to a game. Once started, the standard view appears, which is a bird-eye view on the empty environment. Using the controls it is either possible to add individual modules and build a CPPS from scratch or to import pre-designed facilities. Figure 1A shows such a pre-designed facility consisting of an input-module, which generates the products, a milling module and a painting module which alter the product, a sorting module which can sort product according to color, and a storage module. These are connected by conveyor belts. Additionally to the birds-eye view, the simulation can also be viewed from the front and the back.

A number of modules has already been implemented in FliPSi.

For connecting modules, transport, storage and input/output of products:

Conveyor Belt	Rotary Table
Articulated Robot	Input Module
Output Module	Sorting Module
Storage Module	Color Sensor
Action Sensor	Barrier

For the processing of products:

Milling Module	Drilling Module
CNC Machine	Painting Module

Further modules can be implemented in Unity, using the built-in programming language, or can be imported from external sources, such as Digital Content Creation Tools. It is also possible to import CAD data files, however, since these do not contain information on the behavior and movement of the module or on the effect it will have on the product, these features need to be programmed in Unity.

Once the model has been assembled in Unity, the simulation can be started. All interactions between modules and products are handled by the physics engine of Unity. Sensors within the model record data during the simulation. The type of data to be recorded can be defined by the user. It is possible to record technical data, such as the speed of the conveyor belts and sensor data, such as the color of a product. Further data points, including data that is not normally available from a real CPPS, can be defined as well. Data is recorded once the user pushes the "Start Data" Button. The collected data is saved as a csv file.

4.2. How FliPSi Solves the Aforementioned Problems

FliPSi allows for the generation of more realistic data sets, as models of CPPS are initialized within a physical environment and the interactions between products and modules are simulated using realistic physics. The physics engine is already implemeted in Unity and aspects such as gravity do not need to be implemented. Compared with functional modeling of CPPS this enables the simulation of additional unforeseen faults in a complexity that exceeds functional simulations, e.g. a warping in a single role of a conveyor-belt can cause its blocking by unevenly aligned products several stations or curves later (cf. Figure 1 D). Therefore, FliPSi allows the combination of functional and design simulation to generate large, diverse data sets.

Within FliPSi faults can be manually programmed and automatically injected in a controlled way, such as a sudden stop or change of speed in a conveyor belt, faults in the processing of products or loss of power in the facility. Furthermore, through realistic physical simulation, unforeseen faults can be included in the data set, such as the geometry of a product causing it to get stuck in the production line. Such faults can only occur when the simulation is capable of combining functional and design aspects. Through the physics engine, interactions and the faults that occur through them are handled. The combination of functional and design modeling also allows the simultaneous simulation of multiple different types of fault cases, e.g. a product becoming stuck on a conveyor belt and the loss of power in a processing module.

Another advantage of using game engines for simulation of CPPS is the visualization of the simulation. This is already true during the modeling phase, because modules and facilities are modeled visually in the game engine and any changes can be directly observed in the 3D model of the system. Likewise, the effect of faults, whether injected on purpose or arising from the connection between modules and products, can be visually observed during the simulation, making the process more intuitive.

The simulation environment in Unity also allows for the automatic introduction and consideration of complex relationships between variables within simulation models. By nature

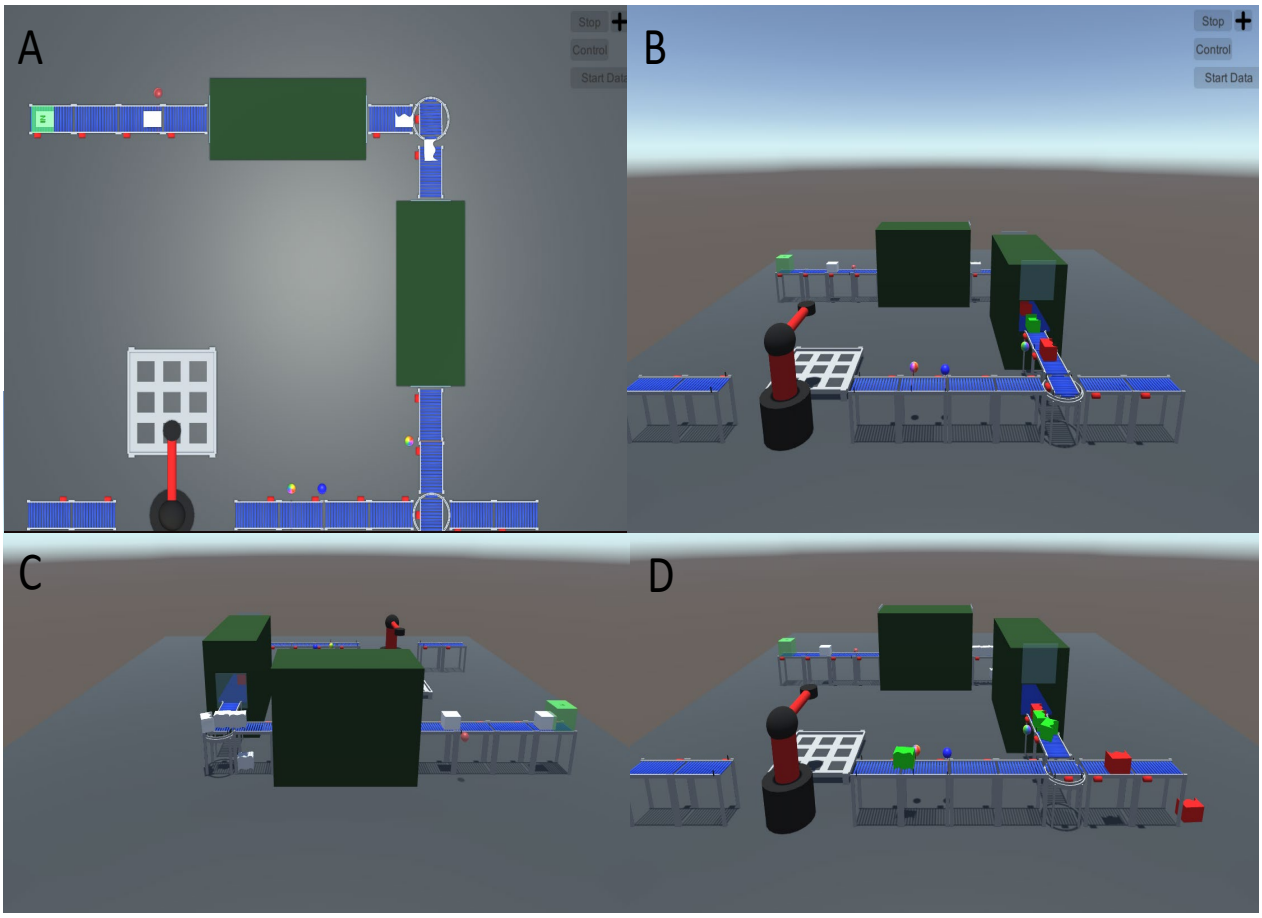

Figure 1. Different views of a simulation in FliPSi. (A) shows the bird-eye view of the simulation and product can be seen on conveyors between the stations. The model of the CPPS consists of an input station, which generates the product, a milling station and a painting station changing the product, a sorting station, a storage and conveyors connecting the stations. (B) shows the front view of the simulation, product can be seen leaving the painting station. (C) shows a fault case caused by a malfunctioning conveyor and (D) shows a fault case in which the product became stuck on the conveyor.

of this setup a complete simulation of rigid body physics is included within every simulation run, enabling the simulation and detection of complex dependencies from a single component up to overarching dependencies over multiple components within the CPPS. This allows for the generation of more detailed and realistic data sets, while keeping the effort of initial modeling reasonable. Because of its ability to generate more realistic data sets, which contain combinations of different fault cases, FliPSi can be employed to generate data sets for the (pre-)training and testing of ML algorithms.

As FliPSi is an open source platform, it offers complete explainability and accessibility. Compared to closed source applications, users of FliPSi can retrace all dependencies and behavior within a simulation run. Additionally, the users can implement their own models alongside the existing models based on their own requirements on complexity and detail. Thus, even specific scenarios can be modeled in an explainable manner, enabling the generation of realistic data sets of CPPS for (pre-)training of ML models in the diagnosis domain.

Furthermore, FliPSi can also be employed to test newly developed and trained ML and diagnosis algorithms. The simulation can be run and fault cases can be injected, which then need to be identified and, possibly, handled by the algorithm. Based on the modularity of FliPSi, data sets for testing asymptotic behavior of ML models can be easily assembled. To achieve this, more and more complex models are created and simulated and the data sets are then used to prove that the ML algorithm shows an asymptotic behavior.

The features describes above additionally allow to use FliPSi for educational purposes. The simulation can be used to showcase the interdependencies of components that are distinctive to CPPS. Since FliPSi is designed in Unity it is easy to understand the implementation and modify or expand it. This makes it a good environment to create challenges for students. As a learning environment it can be used to teach

Table 1. Examples of data points which can be exported from the FliPSi simulation

Module	Exportable Value	Format of Data Point	Possible Further Values (To be added)
Conveyor	Product on conveyor	Yes/No (0/1)	Power consumption Multiple products on conveyor
	Speed of conveyor	Integer	
	Angle of conveyor movement	Integer	
Input Module	Generation of product	Yes/No (0/1)	Time to next product
Color Sensor	Color of Product	Name of Color	Color in RGB code
Drilling Module	Product in module	Yes/No (0/1)	Position of drill hole Material use
	Speed of conveyor inside module	Integer	
	Hole drilled	Yes/No (0/1)	

both about the nature and challenges of CPPS as well as about Anomaly Detection and other applications of ML.

4.3. Functionalities to be added

The physical game engine offers the possibility to simulate scenarios as close to reality as possible. With the help of simulated representations of machines and their associated scripts, which collect process and sensor data, a highly versatile modularity is created. Currently, modules can be added manually and connected to the CPPS. The modules, which are available at this point (see above), can be flexibly exchanged, moved or replaced. The required modules (e.g. drilling module) can be imported, aligned and located in the scenario via a simplified user interface. The data is collected transparently and can afterwards be processed as required. The large number of variants makes it possible to emulate CPPS of any kind and to generate data sets that are normally difficult or impossible to access by industrial companies. It is planned that CPPS can be created automatically with FliPSi in the future. This means a reduction of the manual effort for the compilation of CPPS, higher data set numbers and the achievement of faster results. The basis for this is the specification of the product to be manufactured, the machines to be used and the available quantity of resources (e.g. number per machine). With the help of ML applications, such as supervised learning methods, an optimal plan can be generated from this information, in which order the machines should be arranged and the products manufactured. This data and the data generated during the simulation are saved and made transparently available to the user. To achieve a better concordance with reality, a model fitting will be implemented for models of existing CPPS. Time series data from real CPPS can be used to fit the parameters in the simulation. Thereby, better models of real CPPS can be designed and the data can be used to simulate faults in the real system.

5. Discussion

In summary, a clear trend can be seen regarding the usage of game engines as training ground for different real world problems like autonomous driving or robot behaviour. Although the methods share the basic concept of using a game engine as a tool for real world approximation by creating feasible scenarios and mechanical simulations, the domain for its applications may differ fundamentally.
(Rong et al.,2020) used such an approach for the simulation of self driving vehicle mechanics and as training environment. Here, the focus lies in the functionality of the vehicle as a whole in relation to the outside environment. In comparison, FliPSi's central characteristic lies in the interaction of different mechanical modules and the impact of such interactions. Since (Lee, Hu, Lim, 2021) used game engine tools for the simulation of the environment to train robots and as a benchmark for deep reinforcement learning algorithms, there has been a clear focus on the perfection of singular modules and their tasks. Although in the current state there is a limited number of modules, FliPSi can be seen as a generalization. Robots like in (Lee, Hu, Lim, 2021) could be reimplemented as an additional module to test the complex interaction with the environment, different products and even other modules.
Here, the possibilities and advantages of FliPSi are already recognizable especially regarding its accessibility. As an open source development a collective improvement is possible even with extensions from a variety of different use cases with distinct work fields. The nature of the engine itself grants a basic compatibility. Through the combination of functional and design aspects it is possible to generate complex error scenarios derived from the interaction of both of those aspects. This scenario solves the problem of limited fault data sets. The data sets are then suitable for ML training purposes. By constructing a model of a real CPPS within FliPSi, data sets for pre-training of algorithms can be generated. Using the simulated data, a pre-training task of identifying anomalies and finding the root cause of these can be created. This way, less data of the real system is needed to train the ML algorithm. Modeling alone is not capable of achieving the task of monitoring the state of a CPPS. Models can generate data and can be used to simulate the behavior in case of fault. Models created in FliPSi are even capable of simulating unexpected or rarely occurring faults. How-

ever, even if these faults can be modeled and simulated, they still need to be recognized once they occur in the real system. This task is achieved by using ML algorithms, which can analyze the vast amount of data generated by a CPPS and recognize faulty behavior. With FliPSi we provide are developing a simulation platform to generate better training sets for ML algorithms. The flip side of generating multi-faceted data sets is the need for the creation of complex modules and adequate linking of those. Nonetheless the expected results derived from the combination and interaction between multiple faults, errors and their consequences are visualized and well documented with given data exports.
In summary, FliPSi's application and expansion options far outweigh current limitations and obstacles.

6. Conclusion

In this paper the simulation environment FliPSi is presented. With it, realistic CPPS data can be flexibly generated, which can be particularly beneficial in the diagnostic area of ML. The current use of ML algorithms in the field of Anomaly Detection (RQ1) mainly refers to the use of unsupervised or supervised learning methods, such as classification or decision trees. In the field of diagnosis, the authors are not aware of any algorithm for CPPS so far. The currently existing real data sets often show deficits (RQ2). In particular, the missing data of fault conditions, the challenge in obtaining such data sets, and the aspect that such data is difficult to manipulate in reality pose challenges to the creation of diagnostic algorithms. FliPSi was created in a physical simulation environment whose data generation is transparent and can be handled flexibly. Modular CPPS can be created, whose fault states can be implemented variably. Due to the fact that a physical simulation environment is used, the data is close to reality and adaptable to specific circumstances (RQ3). In the future, the generated realistic data can be used to train diagnostic algorithms and detect anomalies in CPPS. Thus, research gaps in these areas can be closed. It can also serve as a learning environment to teach about CPPS as well as ML. FliPSi will also be made publicly available to interested parties on the institute's website in the future.

References

Angelopoulos, A., Michailidis, E. T., Nomikos, N., Trakadas, P., Hatziefremidis, A., Voliotis, S., & Zahariadis, T. (2019). Tackling faults in the industry 4.0 era—a survey of machine-learning solutions and key aspects. *Sensors*, *20*(1), 109.

Bangsow, S. (2020). *Tecnomatix plant simulation*. Springer.

Bunte, A., Stein, B., & Niggemann, O. (2019). Model-based diagnosis for cyber-physical production systems based on machine learning and residual-based diagnosis models. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 2727–2735).

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 1–58.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017, 13–15 Nov). CARLA: An open urban driving simulator. In S. Levine, V. Vanhoucke, & K. Goldberg (Eds.), *Proceedings of the 1st annual conference on robot learning* (Vol. 78, pp. 1–16). PMLR.

Ehrhardt, J., Ramonat, M., Heesch, R., Balzereit, K., Diedrich, A., & Niggemann, O. (2022). An ai benchmark for diagnosis, reconfiguration planning. In *2022 27th ieee international conference on emerging technologies and factory automation (etfa2022).*

Eiteneuer, B., Hranisavljevic, N., & Niggemann, O. (2019). Dimensionality reduction and anomaly detection for cpps data using autoencoder. In *2019 ieee international conference on industrial technology (icit)* (pp. 1286–1292).

Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., & Broman, D. (2005). The openmodelica modeling, simulation, and development environment. In *46th conference on simulation and modelling of the scandinavian simulation society (sims2005), trondheim, norway, october 13-14, 2005.*

Gertler, J. J. (2017). *Fault detection and diagnosis in engineering systems*. CRC press.

Gómez, Á. L. P., Maimó, L. F., Celdran, A. H., Clemente, F. J. G., Sarmiento, C. C., Masa, C. J. D. C., & Nistal, R. M. (2019). On the generation of anomaly detection datasets in industrial control systems. *IEEE Access*, *7*, 177460–177473.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2). Springer.

Kaparthi, S., & Bumblauskas, D. (2020). Designing predictive maintenance systems using decision tree-based machine learning techniques. *International Journal of Quality & Reliability Management*.

Khan, S., & Yairi, T. (2018). A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing*, *107*, 241–265.

Kleijnen, J. P. (1995). Verification and validation of simulation models. *European journal of operational research*, *82*(1), 145–162.

Krauß, J., Dorißen, J., Mende, H., Frye, M., & Schmitt, R. H. (2019). Machine learning and artificial intelligence in production: Application areas and publicly available data sets. In J. P. Wulfsberg, W. Hintze, & B.-A. Behrens (Eds.), *Production at the leading*

edge of technology (pp. 493–501). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lee, Y., Hu, E. S., & Lim, J. J. (2021). Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 ieee international conference on robotics and automation (icra)* (pp. 6343–6349).

Lindemann, B., Fesenmayr, F., Jazdi, N., & Weyrich, M. (2019). Anomaly detection in discrete manufacturing using self-learning approaches. *Procedia CIRP*, *79*, 313–318.

Liu, J., Guo, J., Orlik, P., Shibata, M., Nakahara, D., Mii, S., & Takáč, M. (2018). Anomaly detection in manufacturing systems using structured neural networks. In *2018 13th world congress on intelligent control and automation (wcica)* (pp. 175–180).

Miller, S., & Wendlandt, J. (2010). Real-time simulation of physical systems using simscape. *MATLAB News and Notes*, 1–13.

Monostori, L. (2014). Cyber-physical production systems: Roots, expectations and r&d challenges. *Procedia Cirp*, *17*, 9–13.

Pang, G., Shen, C., Cao, L., & Hengel, A. V. D. (2021). Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)*, *54*(2), 1–38.

Pittino, F., Puggl, M., Moldaschl, T., & Hirschl, C. (2020). Automatic anomaly detection on in-production manufacturing machines using statistical learning methods. *Sensors*, *20*(8), 2344.

Quatrini, E., Costantino, F., Di Gravio, G., & Patriarca, R. (2020). Machine learning for anomaly detection and process phase classification to improve safety and maintenance activities. *Journal of Manufacturing Systems*, *56*, 117–132.

Rai, R., & Sahu, C. K. (2020). Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. *IEEE Access*, *8*, 71050–71073.

Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., ... Kim, S. (2020). Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 ieee 23rd international conference on intelligent transportation systems (itsc)* (p. 1-6). doi: 10.1109/ITSC45102.2020.9294422

Shaha, M., & Pawar, M. (2018). Transfer learning for image classification. In *2018 second international conference on electronics, communication and aerospace technology (iceca)* (pp. 656–660).

Shao, S., Sun, W., Wang, P., Gao, R. X., & Yan, R. (2016). Learning features from vibration signals for induction motor fault diagnosis. In *2016 international symposium on flexible automation (isfa)* (pp. 71–76).

Torrey, L., & Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques* (pp. 242–264). IGI global.

von Rueden, L., Mayer, S., Sifa, R., Bauckhage, C., & Garcke, J. (2020). Combining machine learning and simulation to a hybrid modelling approach: Current and future directions. In M. R. Berthold, A. Feelders, & G. Krempl (Eds.), *Advances in intelligent data analysis xviii* (pp. 548–560). Cham: Springer International Publishing.

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.