Generation of a Diagnosis Model for Hybrid-Electric Vehicles using Machine Learning

Simon Meckel, Roman Obermaisser, Jie-Uei Yang
Embedded Systems Group, University of Siegen, Germany

***Abstract*—Online fault-diagnosis on system level for complex mechatronic systems takes multiple sensor measurements of the various components into account and contributes to a significantly increased system reliability by tracking down faults in the system at run time, enabling fault-specific recovery actions, such as reconfigurations. Ongoing efforts in the technological development of automobiles, especially in the field of driver assistance systems, yield more and more safety-critical systems, e.g., breaking control systems, and thus generate a high demand for reliable online diagnosis systems. In order to perform fault diagnosis on system level, the interrelations between all measurements must be determined, which is a challenging and often demanding task done by human system experts. In this paper we present a systematic approach based on machine learning to establish an online diagnosis system for a hybrid-electric vehicle model.**

***Keywords*—System Level Online-Diagnosis; Real-Time; Machine Learning; Feature Extraction**

I. Introduction

Modern mechatronic systems, ranging from automobiles with advanced driver assistance systems to aircrafts and power plants, have complex architectures and achieve their functionalities through an interaction of multiple components. Since faults in the system or in a component cannot be prevented at all times, it is the goal of online fault diagnosis to detect, identify and, if applicable, make the system to recover from occurred faults by performing fault-specific recovery actions, e.g., reconfigurations. Typical faults include a faulty design of a component or a system, transient and permanent hardware faults, imprecise specifications or erroneous user operations. Still, online fault diagnosis is often limited to the observation of individual component behavior with specialized diagnosis methods, such as limit checking of sensor measurements or parity checks of system models. Detected discrepancies are straightforwardly mapped to a root cause. Due to the interactions of the components, however, the root causes of faults can naturally be tracked down more precisely when multiple measurements are taken into account. A sound decision on occurred faults builds the foundation for fault-specific recovery actions. Online fault-diagnosis on *system level* aims at detecting and diagnosing faults in a system at run time by deriving and evaluating a multitude of diagnostic features, e.g., from sensor measurements, signal models, or process models. For this, on the one hand the diagnostic procedures for the single components are to be defined. On the other hand, the interrelations of the components are to be determined in order to know how components influence measurements at other components. For a fast and assured root cause analysis it is of high interest in which order the signals are best to be evaluated, to keep the overall number of diagnostic steps as low as possible. The trend in the automotive industry goes towards electric vehicles and hybrid-electric vehicles (HEV). When implementing (safety-critical) driver assistance systems to these newly developed automotives, it is of high interest to include online diagnosis systems as well, which, however, is challenging.

Diagnosis systems are naturally application specific, e.g., model-based diagnosis (in [8] for electric drives) relies on accurate mathematical models of the applications. In [2] we find a robust fault diagnosis technique for the traction system of an electric vehicle. General diagnosis approaches are to be modified and adapted to each system or application, which is often a demanding task and requires the knowledge of human system experts. For that reason there is a high demand for state-of-the-art online diagnosis systems that can be set up in an automatic, systematic, and less effortful manner. Preventing false alarms and no fault found situations as well as diagnosing faults with a broad diagnostic coverage in bounded time are major challenges for online fault diagnosis systems.

In this regard, data-based diagnosis methods have been applied by many researchers (e.g., [3], [4], [11]) as these methods do not require any knowledge about the process parameters or a model about the system. For a feature extraction and fault classification they only require a database of healthy and faulty system conditions. The data-based methods can be additionally combined with explicit knowledge about the process, e.g., human expertise can help to define suitable datasets, diagnostic features, or provide other constraints. Prototypic platforms or implementations that come along with a variety of different sensor measurements, from which the system behavior can be extracted, are highly valuable during the development phase of state-of-the-art hybrid-electric systems (see also [7]). Especially since a faulty system behavior can be systematically simulated with these implementations, they build a foundation for an automated generation of an online diagnosis system.

To cope with the difficulty of setting up a diagnostic model supporting system-level online diagnosis we present a two-stage approach for a systematic design of a qualified diagnostic model for hybrid-electric vehicles. The two-stage approach comprises an offline part and an online part. The offline part includes the selection of diagnostic features, their evaluation strategies, and the extraction of the diagnostic model, i.e., the diagnostic dependencies, with a machine learning algorithm.

The online part refers to the execution of the fault-inference process based on the diagnostic model at run time of the system.

The remainder of this paper is organized as follows: Section II defines the system model, fault model, and the diagnostic model. In Section III we introduce our simulation framework. The applied machine learning algorithm used to generate the diagnostic model is addressed in Section IV, followed by a detailed description of the experimental procedures in Section V. An example and evaluation of our approach is presented in Section VI before the paper is concluded with a future work section and an overall summary in Sections VII and VIII.

II. Model

In order to perform fault diagnosis on system level and to be able to certainly detect and accurately identify faults in a complex system we define a system model, a fault model and a diagnostic model on which our design process is based.

A. System Model

Our goal is to diagnose faults within a mechatronic system, which achieves its functionalities through the interactions of multiple components. We define our system model as generic as possible, yet with a view to hybrid-electric vehicles.

The system model describes the composition of components and their interconnections. In this way, complex tasks can be modeled, e.g., a control system. A component accepts specified types of input signals and produces specified output signals. For instance, the component *electric motor* expects an electric signal input (voltage, current) and produces a mechanical output. Each component may be modeled as a composition of subcomponents, e.g., a battery may comprise multiple cells (subcomponents). The relation between input and output signals of components are either directly defined by mathematical equations or, on a higher hierarchical level, results from the interrelations of subcomponents. In our domain of interest (hybrid-electric systems) components are mainly belonging to the areas of power systems (e.g., motors) and electronics (e.g., sensors). Connections between components are either electrical, mechanical, or a signaling line. Sensors are components that convert physical quantities (e.g., voltage) into data values to be used for processing. With the sensors, signal patters at the input and output ports of components are collected. This data forms the foundation for the system-level analysis.

B. Fault Model

Faults in a system originate from various reasons, e.g., a faulty design of a component, wear, overload, software faults, or erroneous user operations, amongst others. We classify faults according to their kind of appearance, their duration, and their underlying cause. In our fault model a fault can abruptly or incipiently appear, that means, a component or a subsystem can either fail suddenly without any prior indication within measurements or a faulty system behavior can unfold over a certain time span (degrading performance). Once occurred, a fault can be permanent, intermittent, or transient, however, within this work we restrict faults to be permanent.

C. Diagnostic Model

Faults in the system (failure or degradation of a component) are reflected in the measured signals. A diagnostic decision can be consequently conducted based on the analysis and processing of fault indications derived from the sensor signals. In the case of a fault in a component the monitored signal behaviors differ from a no fault situation and the changes can affect multiple signals. Obviously, these characteristic changes may differ depending on the driving situation (e.g., acceleration mode or cruise mode); accordingly, one and the same fault may manifest itself in a different way in the signals.

The diagnosis of different faults requires the execution of different signal processing and feature extraction tasks in a defined order [9]. The dependencies between the diagnostic operations are modeled in a diagnostic directed acyclic graph (DDAG) denoted as $G = (T, E, (\ell_e)_{e \in E})$, where T is a set of tasks t and E is a set of ordered pairs (t, t') modeling a precedence relation between two tasks. Edges from E are called *(logical) channels*. For a channel e, the number $\ell_e \in \mathbb{N}$ specifies the size of the message that is sent via the channel e. Task t' depends directly on t, iff $(t, t') \in E$. A task may depend on multiple others and in turn may work as a prerequisite for other tasks [6].

In case of a fault, the attained fault indications initially provide evidence for a certain fault but may yet hold inconclusive information at a certain node of the DDAG (e.g., if confirmative information from other nodes is not yet available). The degree of confidence of a correct fault identification increases with more information merged. The generation of the diagnostic model, i.e., finding signal relations in the data as well as adapting the diagnostic features is performed by the use of a machine learning algorithm.

III. Simulation Framework

The simulation framework is the environment for our system model, i.e., an HEV-model, to be executed. The simulations of healthy and faulty system behavior together with the process of gathering sensor data produces the inputs for the subsequent automatic diagnostic model generation.

A. Hybrid-Electric Vehicle Model

In order to demonstrate our two-stage diagnosis system approach for hybrid-electric vehicles we introduce a Simulink model[1], that offers an abstraction of a hybrid-electric car.

As depicted in Figure 1 the model integrates an electrical and mechanical part with various components. The main components of the high-voltage electrical part include an electric motor, a generator, a voltage converter and a battery. The mechanical part comprises a power split device that combines

[1]Hybrid-Electric Vehicle Model in Simulink, http://www.mathworks.com/matlabcentral/fileexchange/28441-hybrid-electric-vehicle-model-in-simulink

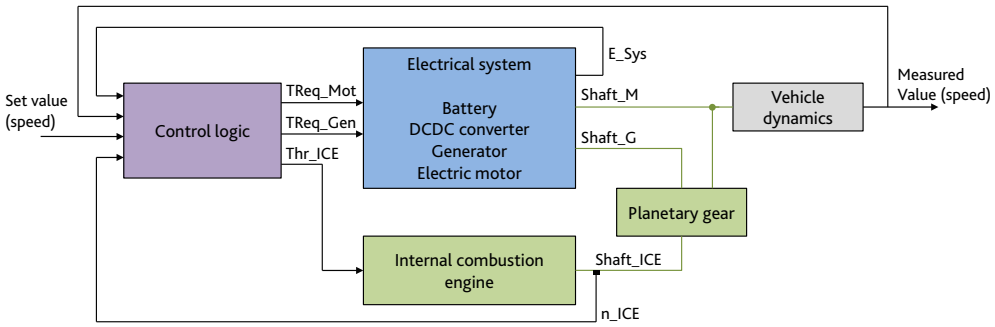

Fig. 1. Overview of the HEV-model

an internal combustion engine (ICE) with the generator and the driveshaft, while the electric motor is directly connected to the driveshaft. Via the power split device the ICE fulfills two tasks: supporting the electric motor to drive the car and extending the car's operating range by charging the battery via the generator. A mode logic manages the interaction between these units (e.g., turning on or off the generator depending on the current battery state of charge and driving situation). Many built-in sensors, such as voltage sensors, current sensors, torque sensors, or tachometers, allow to capture the components in various driving situations and health states. The temperature of different components as well as the mode logic is also monitored. Connections between the components can be mechanical (green) or electrical (blue, within the subsystem). A connection via a black line indicates that control signals are being exchanged. The Simulink model allows to simulate the car's behavior according to a driving cycle (i.e., a velocity profile) input.

B. Fault-Injection

In order to simulate different faults in the components, e.g., in the battery, electric motor, or generator, we modified and extended the original Simulink model. Fault types include complete component failures that suddenly occur (step-wise faults) as well as scenarios where components show a degrading functionality over some time (drift-wise faults). The injection of a fault can be done at different levels of detail. For instance, if the battery of our HEV-model is modeled as a composition of multiple cells, each cell can fail independently (e.g., voltage drop, short circuit). Accordingly, specialized procedures are implemented to accomplish the fault injection for each relevant component. For every component the underlying model can be exchanged or modified depending on the purpose or the requirements of the analyses to be conducted. It can be generalized, the more different faults can be simulated, the greater the diagnostic coverage that can be achieved as more datasets are available for the generation of the diagnostic model. Faults can be induced at run time of the simulation at arbitrary times, independent from the current driving situation. In this way we established a fault-injection framework[2] to produce various datasets with the monitored signals of fault-free and faulty system behaviors. This data implicitly holds the information regarding fault effects and component (resp., signal) interrelations and forms the basis for the machine learning algorithm to extract the diagnostic dependencies.

[2]Extended HEV-model with fault injection framework, https://networked-embedded.de/es/index.php/dakodis.html

IV. Machine-Learning

The performance of diagnosis systems is often limited where particularly two factors play an important role. Firstly, fault diagnosis typically starts at component level, that means, single components of a system are examined individually. Measurements or process outputs of such a component or a subsystem are not seen in the context of others and faults are straightforwardly mapped to root causes. It is thereby already challenging to select and adjust an appropriate diagnostic method. This also includes the selection of diagnostic features and their evaluation process, which is often done manually and necessitates experienced human system experts. Secondly, modern systems can be greatly complex and extremely challenging to diagnose when measurements are not seen in context, i.e., on system level. In order to isolate and manage an occurred fault, its root cause has to be identified through a process of evaluating confirmative and unsupportive diagnostic information from various measurements. During this process particular faults are included or excluded and the actually occurred fault is eventually specified.

A. Selection of Fault Diagnosis and Machine Learning Models

A DDAG as introduced in Section II-C has the natural structure of node dependencies, which creates a sequential implementation of tasks to be performed for the diagnostic process on system level. We intend the creation of DDAGs to be a standardized process, that avoids human subjective influences. Therefore, data-driven methods are proposed for the creation of DDAGs; the advantages include the following points:

- Data is taken from the process and measurements only. The algorithm relies less on human expert knowledge and experiences and thus reduces personnel costs and potential human conflicts.
- The premises of knowledge of the internal structures and logics of systems are excluded. This trait is a huge advantage in the industry for fault diagnosis.

- With the predictor importance estimate good indications of which sensor data are important and which are not is provided.
- More flexibility and better maintainability is provided when system topologies are altered or new faults are to be analyzed.

Machine learning algorithms interpret and predict datasets and can support the generation of a data-driven mode. The classification decision tree model with standard CART (Classification and regression trees) algorithm is selected for our research for the following reasons:

- It handles multi-class classifications with less effort.
- Not all features for the test data have to be present and analyzed for classification.
- A decision tree can be converted to a DDAG.
- The sequential tasks presented by a decision tree are straightforwardly to be scheduled to a given system infrastructure.

B. Decision Tree Algorithm

The learning process requires input datasets, e.g., measurements of the system, preferably for all scenarios to be classified. Besides, each dataset must have a label, indicating to which fault class it belongs. Another prerequisite is the definition of a pool of diagnostic features potentially to be applied for the diagnostic inference process.

In the field of machine learning the features in datasets are also referred to as predictors. Standard CART selects the split predictor that maximizes the split-criterion gain over all possible splits of all predictors [1]. Therefore, the calculation of Gini's Diversity Index (GDI, see also Gini Impurity) is included into the algorithm.

$$I(p) = \sum_{i=1}^{J} p_i(1-p_i) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = 1 - \sum_{i=1}^{J} p_i^2 \quad (1)$$

The mathematical expression of GDI is shown in Equation 1; p_i is the probability of the label i being classified into one sub-node. The term $(1-p_i)$ consequently denotes the probability of misclassified labels. A node containing only one type of labels has a GDI of zero and an impure node will have a positive GDI. Assuming a predictor A splits S into subsets S_i the Gini gain can be calculated from the weighted average over all sets resulting from the split (i.e., the average entropy) and the entropy of the original set:

$$GiniGain(A,S) = Gini(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} Gini(S_i) \quad (2)$$

In Equation 2 $GiniGain(A,S)$ denotes the Gini gain after the split with predictor A, S_i as the partition count in sub-node i and S as the count before the split. After evaluating the Gini gain for every predictor, the algorithm selects the predictor which has the highest Gini Gain for splitting. The calculation iterates itself as the tree continues to grow. It is terminated when the termination condition is met.

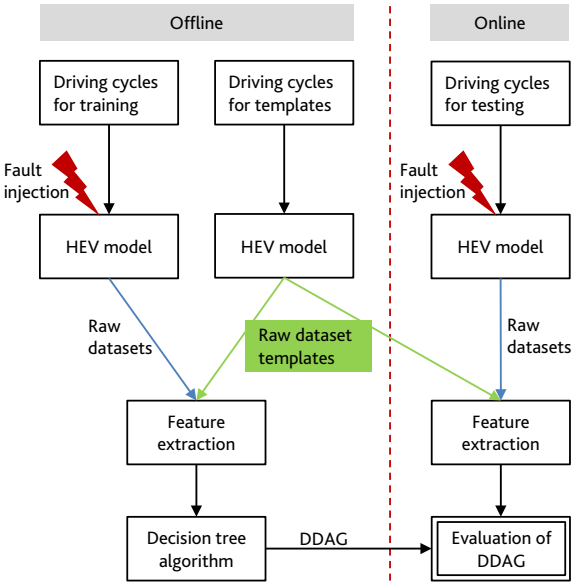

Fig. 2. Generation of the diagnostic directed acyclic graph

V. Experimental Procedure

A valuable output of the machine learning algorithm in the form of a decision tree can be only expected, if certain prerequisites are fulfilled. They include a suitable selection of diagnostic features and datasets of healthy and faulty system conditions. The decision tree then reveals the diagnostic dependencies, i.e., the order of diagnostic conclusions (split predictors and corresponding split parameters). The tree can be converted into a DDAG, which implicitly contains information about necessary input signals, such as sensor data or process parameters, and incorporates the online processing steps for the feature extraction, summarized in the form of diagnostic tasks.

A. Process Overview

The experimental procedure is outlined in the schematic diagram in Figure 2. The *offline* part is shown to the left of the dashed line and describes the generation process of the diagnostic dependency graph. Once the graph, i.e., the logical model, is established it can be scheduled to the physical system (e.g., ECUs or other computation units), forming the *online* diagnosis system.

Simulating the HEV-model with specially designed driving cycles for so called *templates* and for training yields raw datasets from which features are extracted (Sections V-B and V-C). The obtained training data (healthy and faulty datasets) is fed to the decision tree algorithm to generate a decision tree that classifies the training data. The tree is evaluated by cross-checking the prediction of test data and its labels. The growing of a decision tree ends when the termination condition is met. This is dependent on the correlation of data, which is highly unpredictable. The overall time of creating a decision tree can be hardly estimated, however, after the tree is determined all calculation procedures are plannable and predictable. The

decision tree is transformed into a DDAG, which includes additional information about the signals to be evaluated at a certain stage and can combine several processing steps, e.g., subsequent decisions that rely on the same features. The implementation of a DDAG with test data is regarded as an online process. This implies that an execution of a DDAG with test data or new data in a real-time environment is possible.

B. Driving Cycles for Templates, Training, and Testing

The creation of driving cycles in combination with simulations of the vehicle behavior is important for two reasons. On the one hand, as further addressed in Section V-C, useful features can be obtained when the norm of the signals is known, i.e., the healthy system behavior. This knowledge is extracted and summarized by conducting simulations of a set of abstracted driving cycles of particular speed and acceleration profiles. We denote the obtained raw signals from these simulations as *templates* in the following. Considering a speed range from $20\,km/h$ to $120\,km/h$ and a velocity interval of $20\,km/h$ while covering the three driving modes cruise, acceleration, and deceleration (all constant), we create $6 \cdot 3 = 18$ simplified velocity profiles all with a duration of at most $40\,s$. Making use of such abstracted driving cycles for the purposes of diagnostic feature extraction as well as training and testing, the classification algorithm implies that complex driving cycles such as the WLTP[3] driving cycles or real world driving situations can be segmented into parts of constant acceleration and speed values. On the other hand, the completeness of driving cycles (situations) is important as the machine learning algorithm cannot predict well in domains not covered by driving cycles. It is therefore essential that both the driving cycles used for the template signals as well as the driving cycles used to generate training data for the machine learning algorithm are appropriately chosen. For instance, the machine cannot predict the behavior of the car driving at $120\,km/h$ well if there was not any prior knowledge about the behavior with inputs of this region.

Driving cycles for training induce random offsets to velocities and accelerations, which assures certain offset tolerances for classifications. The model possibly performs unsatisfactorily on test data with offsets, if training data is perfectly adjusted and contain no offsets between training and templates.

The third set of driving cycles is created in order to test the performance of the classifier. These driving cycles also contain offsets of both parameters and show a good coverage inside the overall velocity range. Besides, driving cycles for testing should not coincide with those used for training.

C. Diagnostic Feature Selection

A good feature captures the characteristics of a system in the current state and a set of multiple good features can differentiate between a system operating in a healthy and in a faulty state [5]. The feature output information is used for the learning model which means that the selection of features directly influences the performance of the machine learning model. Expertise and experiences of human system experts at this stage are required as no universal algorithm for feature selection exists. For example, a good and simple feature is the percentage deviation of a measured value to its norm value. Such a feature can be directly analyzed, e.g., by limit checking. Although it is easy to define such a feature, it is challenging to set the limits for checking. It is also to be expected that these limits have to be adaptive in order to take account for varying process states, e.g., system load.

For our model we define two different kinds of feature selection methods and compare them against each other: *boundary feature* and *percentage feature*. Since both methods require template information (i.e., healthy signal information) to process the features, all necessary data is stored during an initialization phase and can be consequently accessed online when the diagnosis system is operating. The feature selection methods can be generalized for many other processes as well. The first feature selection method is an extension of limit-checking. A fixed-limit checking method or a moving-limit checking method typically outputs an in-bound or out-of-bound result and is too insufficient when aggregated into sets to differentiate between different faults. The extended method categorizes the difference between the real (i.e., measured) signal value and the template value with respect to the template value at the same time step into five groups, which point out the value deviations and hence, allowing a finer quantification. Potential outputs expressing the deviation are then $[-1, -0.5, 0, 0.5, 1]$.

As shown in Equation 3, the second selection method involves the calculation of the error percentage of the measured signal with respect to the template signal.

$$percentage = \frac{real\_signal\_value - template\_value}{template\_value} \quad (3)$$

Using error percentages to depict the signal relationships can be regarded as an extension of a limit-checking method but with very fine-quantified boundaries. With finer quantifications, the ability to tell differences between numbers apart becomes better and this can help the machine learning algorithm to classify data sets into their labels.

Prior to the calculation of the feature, for both methods, the correct template has to be chosen. For this, the current driving situation is to be determined and expressed by an acceleration and velocity value, which are seen to be valid for a limited time interval. Feature Extraction is as follows:

1) Extraction of the acceleration and the initial value of the real velocity profile in the time frame of interest.
2) Selection of the velocity template which resembles the real velocity profile the most.
3) Acquisition of the features by performing the designated feature selection method on all the signals of interest from the real measurements and the templates.

[3]Worldwide Harmonized Light-Duty Vehicles Test Procedure, https://www.unece.org/fileadmin/DAM/trans/doc/2014/wp29/ECE-TRANS-WP29-2014-027e.pdf

In our scenario with simplified driving cycles for the training procedure these steps can be easily performed, however, working with velocity profiles such as the WLTP driving cycles one needs to segment the driving cycle at run time in order to provide the necessary parameters for the template selection. This information must be frequently updated for the online-diagnosis system to operate correctly.

D. Decision Tree Implementation

According to Figure 2 the decision tree is created from the datasets after feature extraction. The "minparent" parameter is set to 1 as the termination condition to ensure full growth of the tree. The output decision tree is likely to be overfitted to the observations. Overfitting occurs when the machine learning model is adjusted too well to training data and therefore could have worse predicting abilities. Two drawbacks arise because of an overly-grown tree. One is the possibility that the accuracy for predicting test data labels decreases; the other is that due to the high number of branches and conditions, it causes a lot of unnecessary efforts to implement for online diagnosis. Pruning is used to trim the tree branches to an extent such that the trade-off between branch number and accuracy of the tree is reasonable for implementation.

Three performance indicators, namely *accuracy*, *cross-validation error*, and *resubstitution error*, are presented in this paper to compare the performance between different trees. Accuracy is calculated from predicted labels of test data from tree and test data labels. A higher accuracy implies that more labels are correctly predicted from the tree and therefore the tree is a more desired result. Cross-validation error is obtained by bagging training data into ten groups, nine of which are used to create a tree and the other one is used as test data. The nine groups are selected randomly in every iteration and there are in total ten iterations. Cross-validation error is calculated by averaging misclassified labels in all ten iterations. It provides an indication of how balanced the labels in the training data are. Resubstitution error is calculated from the mismatched labels between training data and predicted labels from training data from tree. A low resubstitution error represents a good ability of the tree with predicting training data labels. With each prune the resubstitution error rises. A low resubstitution error does not guarantee a good accuracy of a tree. In many cases, overfitting may occur.

VI. Examples and Evaluation

We now demonstrate how the signal dependencies for a system level online diagnosis are automatically extracted from raw datasets, derived with the help of our fault-injection framework, according to the procedure described in Section V.

A. Diagnostic Dependencies

Figure 3 shows the outcome of the machine learning algorithm based on 62 training datasets. For the simulation we used the percentage features. The fully grown tree is pruned four times to decrease potential false classifications due to overfitting. In the simplified scenario we assume that only one fault occurs at a time and the potential faults are: failure of the electric motor (label 1), failure of the generator (2), failure of the battery (3), or ICE failure (4). The label (0) indicates that no fault has occurred. The decision tree graphically illustrates which signals have to be analyzed in order to include or exclude particular faults during the decision process. As we see from the applied predictors, at least two signals need to be evaluated to conclude a fault. For instance, a battery failure (label 3) is concluded after examining the "motor current feature" and the "battery current feature". Possibly, this fault is indicated by evaluating just one feature, however, on system level the decision includes multiple features and is generally more precise and valuable.

Figure 4 indicates the predictor importance estimate for the decision tree. The most significant predictors are x1 and x2, which is in accordance with the top-level decisions of the tree. It seems counterintuitive at first that the vehicle speed (predictor x4) is not evaluated, however, it needs to be considered that the speed information is already included in the processing when a template is chosen. The estimate gives insight into the importance of certain sensors and can beneficially influence future modifications to a system, e.g., establish redundancy of particular sensors or discard others.

B. Comparison of Feature Selection Methods and Fault Types

Table I summarizes the characteristic values of several simulation sets and allows a comparison between the two feature selection methods and the different fault types:

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT SIMULATION SETS

Simulation	Accuracy (%)	Cross Validation Error	Resubstitution Error	Prune Level
$A1$	74.603	0.282	0.158	0
$A2$	73.016	0.286	0.256	5
$B1$	87.302	0.139	0.004	0
$B2$	87.302	0.135	0.068	4
$C1$	88.679	0.219	0	0
$C2$	90.566	0.245	0.063	2
$D1$	91.045	0.161	0	0
$D2$	86.567	0.163	0.085	4

A: Fault Diagnosis with templates 20 and boundary features.
B: Fault Diagnosis with templates 20 and percentage features.
C: Predictive Diagnosis with templates 20 and percentage features.
D: Predictive Diagnosis with templates 10 and percentage features.

Comparing the simulation sets A and B we conclude that percentage features outperform boundary features in the fault diagnosis, i.e., determining the root cause of a faulty system behavior. This is due to the fact that percentage features can describe the differences between healthy and faulty signals in more detail. Simulation sets C and D show the algorithm performance for a predictive diagnosis. In this scneario we intend to predict the lifetime of a component when it shows

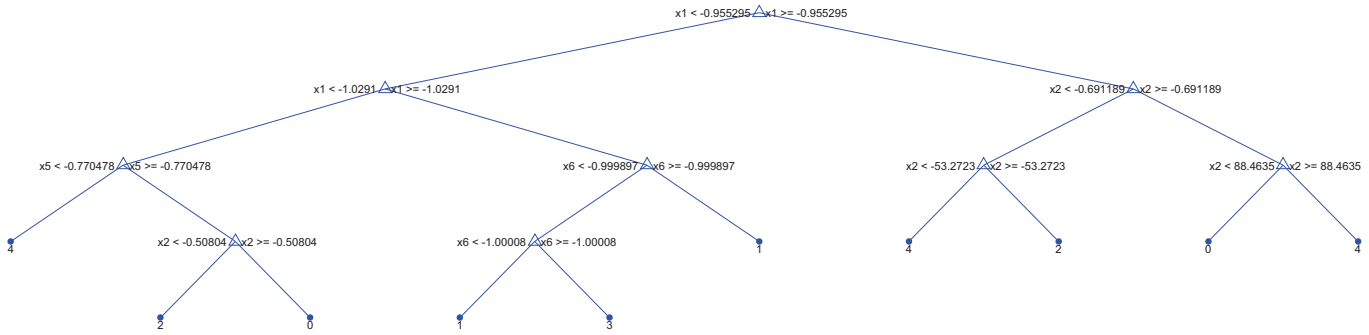

Fig. 3. Decision tree of simulation $B2$ (Table I); predictor representation: x1: Motor current feature, x2: Generator current feature, x3: Engine torque feature, x4: Vehicle speed feature, x5: Generator speed feature, x6: Battery current feature; label representation: 0: No faults, 1: Motor failure, 2: Generator failure, 3: Battery failure, 4: ICE failure

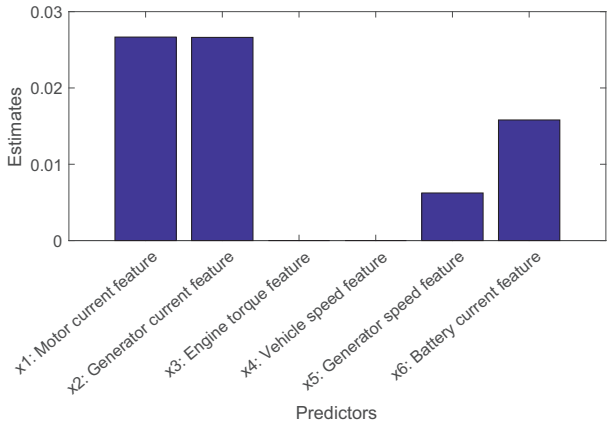

Fig. 4. Predictor importance estimates of simulation $B2$

a degrading performance over time by classifying its current healthy status [10]. These two simulation sets use percentage features and additionally compare the influence of having more healthy raw datasets (templates) available by reducing the velocity intervals to $10\,km/h$ (templates 10; recall Section V-B). For fault diagnosis with templates 20 and percentage features an accuracy of $87.302\,\%$ is achieved (Simulation B). The low cross-validation error of 0.135 shows a good balance of different labels in the training data. The resubstitution error increases to 0.068 after four prunes, yet the accuracy remaines the same. This tree can be utilized as a DAG for fault diagnosis. As for predictive diagnosis, an accuracy of $90.566\,\%$ was obtained using templates 20 and percentage features (Simulation C). A cross-validation error of 0.245 shows a slight imbalance in training data, because healthy labels were the majority among all labels. All simulations using percentage features (B-D) show an accuracy of about $90\,\%$ which is a good result considering the simplicity of the selected features (expressing simple deviations of a signal from its norm) and the little number of diagnostic decisions necessary. Definitely, improved results can be achieved in the future when advanced diagnostic features also handle trend analyses of signals or take account of characteristic signal patterns.

C. Distributed Fault Diagnosis Process

In order to establish an online diagnosis system, which can be implemented on a real system, potentially on a distributed architecture, we need to convert the decision tree into a DDAG according to the diagnostic model introduced in Section II-C. In our simplified example all diagnostic tasks include the two steps feature extraction and feature evaluation. The result of the conversion of the decision tree (Figure 3) into a DDAG is shown in Figure 5. In the DDAG the labels indicate the faults (and no fault situation) that are identified at the corresponding processing steps.

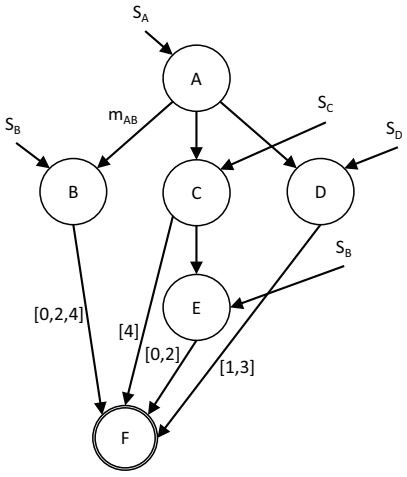

Fig. 5. Diagnostic directed acyclic graph of simulation B2

Table II gives an overview of the diagnostic tasks and necessary input signals. Whenever a feature is evaluated multiple times in a row (e.g., x1 in Figure 3) it becomes one task (compare with node A in Figure 5) since in our model, a diagnostic task is not limited to one processing step, yet, to the evaluation of one feature. To comply with real-time environments, the dataset collection with labels for training, feature extraction, decision tree generation, and pruning are executed offline. The pruned decision tree is then introduced as a DDAG into the real-time environment. In online diagnosis, procedures include signal collection from sensors, determina-

TABLE II
DIAGNOSTIC TASKS AND SIGNALS

Node	Task (feature gen. and eval.)	Signal	Measurement
A	Motor feature 1	S_A	current
B	Generator feature 1	S_B	current
C	Generator feature 2	S_C	speed
D	Battery feature 1	S_D	current
E	Generator feature 1	S_B	current
F	Final fault decision		

tion of the current driving situation, feature extraction, feature processing according to scheduled DDAGs, and eventually the conclusion of a label, which reveals the state of the vehicle in the given time frame. These procedures are repeated throughout the whole driving cycle and real world driving situation, respectively.

A stepwise fault-inference process operating on system level is qualified for complex real world systems. Especially when the system to be diagnosed consists of multiple distributed processing units, not all relevant data may be available for the diagnosis at all times. With our proposed algorithm a diagnostic model, which organizes the successive generation, evaluation, and combination of the relevant information for an online fault-diagnosis process, can be established.

VII. Future Work

In order to prepare the algorithm to also work with real automotive data from project partners in the future, we intend to refine our algorithms by extending both the HEV-model and the according fault-injection possibilities to produce data closer to reality. Besides, the diagnostic features applied so far can be seen as extensions of limit checking. Since faults may leave complex traces within signals, more sophisticated diagnostic features can improve the accuracy of our algorithms. With a library of different feature extraction methods available, an algorithm that automatically selects the best performing features could further automate the design process. We also intend to compare and validate our machine learning model with related methods of the field of artificial intelligence. In terms of time guarantees for the online fault-diagnosis process a tool for timing analyses of all operations of the process is to be established, in order to estimate their worst case execution times.

VIII. Conclusion

In this paper, we motivate online diagnosis to be performed on system level. Naturally, faults in a complex system can be tracked down more precisely with more (sensor or process) data taken into account for the diagnostic inference process. We introduced a hybrid-electric vehicle model and an associated fault-injection framework. With the ability to simulate the health status (e.g., failures or degrading performance) of the vehicle components in different driving situations we established a database and the applied machine learning algorithm extracts signal interrelations and feature evaluation estimates. The result of the algorithm, a decision tree, is converted into a diagnostic directed acyclic graph. That way, the obtained diagnostic model can be executed at run time on a given physical infrastructure, e.g., at distributed computation nodes of a hybrid-electric vehicle, thus realizing an online diagnosis system operating on system level. Our automatic and systematic approach relies less on human expert knowledge and experiences regarding component interrelations and can consequently help to reduce personnel costs and potential human conflicts. Simulations comparing different fault types and feature selection methods show that the proposed algorithm achieves an accuracy of over $90\,\%$ (correct fault classifications) with a very compact DDAG, that means, only few decisions are necessary. Considering the high level of automation of the approach and the relatively simple diagnostic features that were applied this is a notable result. Yet, there is room for improvements as diagnosis systems for safety-critical applications typically require a higher accuracy in terms of a correct fault identification.

Acknowledgment

This work was supported by the DFG research grants LO748/11-1 and OB384/5-1.

References

[1] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[2] Mohand Arab Djeziri, Rochdi Merzouki, and Belkacem Ould Bouamama. Robust monitoring of an electric vehicle with structured and unstructured uncertainties. *IEEE transactions on vehicular technology*, 58(9):4710–4719, 2009.

[3] Hong Guo, Jacob A Crossman, Yi Lu Murphey, and Mark Coleman. Automotive signal diagnostics using wavelets and machine learning. *IEEE transactions on vehicular technology*, 49(5):1650–1662, 2000.

[4] Humberto Henao, Gerard-Andre Capolino, Manes Fernandez-Cabanas, Fiorenzo Filippetti, Claudio Bruzzese, Elias Strangas, Remus Pusca, Jorge Estima, Martin Riera-Guasp, and Shahin Hedayati-Kia. Trends in fault diagnosis for electrical machines: A review of diagnostic techniques. *IEEE industrial electronics magazine*, 8(2):31–42, 2014.

[5] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.

[6] Seungbum Jo, Markus Lohrey, Damian Ludwig, Simon Meckel, Roman Obermaisser, and Simon Plasger. An architecture for online-diagnosis systems supporting compressed communication. In *Digital System Design (DSD), 2017 Euromicro Conference on*, pages 62–69. IEEE, 2017.

[7] Jianhui Luo, Fang Tu, Mohammad Shafiul Azam, Krishna R Pattipati, Peter K Willett, Liu Qiao, and Masayuki Kawamoto. Intelligent model-based diagnostics for vehicle health management. In *System Diagnosis and Prognosis: Security and Condition Monitoring Issues III*, volume 5107, pages 13–27. International Society for Optics and Photonics, 2003.

[8] Yi Lu Murphey, M Abul Masrur, ZhiHang Chen, and Baifang Zhang. Model-based fault diagnosis in electric drives using machine learning. *IEEE/ASME Transactions On Mechatronics*, 11(3):290–303, 2006.

[9] Roman Obermaisser, Rubaiyat Islam Sadat, and Fabian Weber. Active diagnosis in distributed embedded systems based on the time-triggered execution of semantic web queries. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*, pages 222–229. IEEE, 2014.

[10] Rune Prytz, Slawomir Nowaczyk, Thorsteinn Rögnvaldsson, and Stefan Byttner. Analysis of truck compressor failures based on logged vehicle data. In *Proceedings of the International Conference on Data Mining (DMIN)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[11] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.