Online diagnosis of accidental faults for real-time embedded systems using a hidden Markov model

Ning Ge, Shin Nakajima, Marc Pantel

To cite this version:

Ning Ge, Shin Nakajima, Marc Pantel. Online diagnosis of accidental faults for real-time embedded systems using a hidden Markov model. SIMULATION, SAGE Publications, 2015, vol. 91 (n 10), pp. 851-868. <hal-01316834>

HAL Id: hal-01316834

https://hal.archives-ouvertes.fr/hal-01316834

Submitted on 17 May 2016

Online diagnosis of accidental faults for real-time embedded systems using a hidden Markov model

Ning Ge[1], Shin Nakajima[2] and Marc Pantel[1]

Abstract

This article proposes an approach for the online analysis of accidental faults for real-time embedded systems using hidden Markov models (HMMs). By introducing reasonable and appropriate abstraction of complex systems, HMMs are used to describe the healthy or faulty states of system's hardware components. They are parametrized to statistically simulate the real system's behavior. As it is not easy to obtain rich accidental fault data from a system, the Baum–Welch algorithm cannot be employed here to train the parameters in HMMs. Inspired by the principles of fault tree analysis and the maximum entropy in Bayesian probability theory, we propose to compute the failure propagation distribution to estimate the parameters in HMMs and to adapt the parameters using a backward algorithm. The parameterized HMMs are then used to online diagnose accidental faults using a vote algorithm integrated with a low-pass filter. We design a specific test bed to analyze the sensitivity, specificity, precision, accuracy and F1-score measures by generating a large amount of test cases. The test results show that the proposed approach is robust, efficient and accurate.

Keywords

Real-time embedded system, simulation, online diagnosis, accidental fault, hidden Markov model

1. Introduction

1.1. Motivation

As the scale and complexity of real-time embedded systems are rapidly increasing due to the growth of functional and non-functional requirements, resource-consuming simulation technologies are not able to detect all faults in systems design. Thus, the use of abstract models to simulate the behavior of complex systems preserving mandatory semantics is now a strong requirement. The efficiency and accuracy of model-based approaches depend on the appropriate abstraction and reasonable assumptions. This article restricts the attention to the problem of online accidental fault diagnosis for monitored real-time embedded systems.

According to the terminology defined in the literature,[1–3] the accidental faults that happen in hardware and software are caused either by design faults, physical wear and tear, environmental conditions or by a peculiar set of inputs/excitations given to the system. They can be both expected or unexpected faults. Although accidental faults are usually very rare in the runtime, still it is mandatory to detect and diagnose them in real-time embedded systems to ensure that the fault tolerance infrastructure will automatically maintain the functionality. This is currently the main method for improving the reliability and dependability of embedded systems.

The detection and isolation (diagnosis) of faults (FDI) in engineering systems has been of great practical significance since the 1990s.[4] FDI is dedicated to monitoring a system, identifying when a fault has occurred, and pinpointing the type of fault and its location. One main FDI methodology derives faults from models, classified into the category of model-based FDI, which relies on an explicit mathematical or knowledge-based model of the controlled system. Various model-based FDI approaches have been widely used in process engineering and dynamic systems, such as observer-based, parity-space, parameter-estimation methods, etc.[5–7]

[1]University of Toulouse, IRIT, Toulouse, France
[2]National Institute of Informatics, Tokyo, Japan

Corresponding author:
Ning Ge, University of Toulouse, IRIT, Toulouse, France.
Email: ning.ge@enseeiht.fr

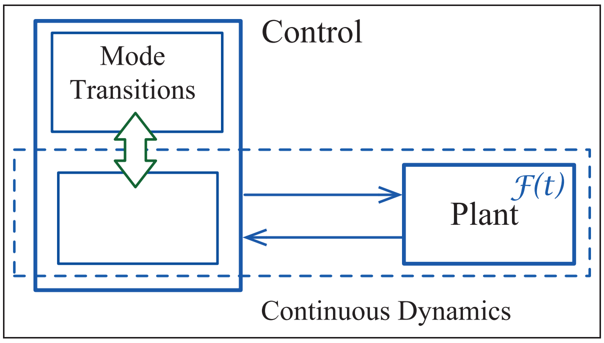

Figure 1. Accidental fault diagnosis in CPS.

1.2. Previous works

The research topic in this work derives from the previous work by Nakajima et al.,[8] which proposed a co-analysis framework for the automated analysis of cyber-physical systems (CPS).[9] As a kind of real-time embedded system with feedback loops from the environment, CPS receive more and more attention from researchers in recent years. CPS introduce a new paradigm to software-intensive systems, in which the controller (system) is strongly affected by the feedback from the plant (environment). In Nakajima et al.,[8] the behavior of Simulink models is taken in an as-is manner determined by the simulation algorithms. The approach combines logic-based formal analysis methods with numerical simulations to enable the analysis of an under-constrained controller design, which cannot be handled by co-simulation. As shown in Figure 1, a controller may have several control modes and evolve by enabling mode transitions. Since some of the Simulink model descriptions represent hardware components, the controller is expected to be able to online diagnose the accidental fault $\mathcal{F}(t)$ occurring in the hardware. Therefore, an open question remaining in Nakajima et al.[8] is the analysis of possible accidental faults in Simulink models in order to increase the level of robustness.

To describe accidental faults in Simulink models that represent hardware components, it is required to define a failure rate parameter for each component (usually very low values). The Simulink models are then able to randomly generate some faulty signals according to these parameters. A modeling method is needed to simulate the system's behavior to observe the output data from the plants and online determine whether the hardware components encounter an accidental fault as well as which component does.

1.3. Contributions

This article targets the accidental fault diagnosis problem for real-time embedded systems by proposing a modeling method based on hidden Markov models (HMMs).[10–12]. In HMMs, the system being modeled is assumed to be a Markov process with unobserved (hidden) states. By analyzing the observation sequences from the hidden states, the state sequence can be deduced. HMMs have thus been widely used in temporal pattern recognition such as speech, handwriting, gesture recognition, etc.

This work introduces reasonable and appropriate abstraction of complex systems relying on HMMs to model the faulty and healthy states of system's components. They are parametrized to statistically simulate the real system's behavior. The challenge in this work consists in limited accidental fault data, because it is not easy to obtain rich accidental fault data from a system. It is not reasonable to rely on the limited amount of faulty data to adapt the parameters in the initial and transition matrices using the Baum–Welch algorithm.[13] We thus propose to fix these two matrices using the failure rate parameters of hardware components while estimate the emission probabilities. The estimation method is based on the principles of fault tree analysis (FTA) and the maximum entropy in Bayesian probability theory. A fault propagation distribution is thus computed, whose parameters are adapted using the backward algorithm and observations. The parameterized HMMs are then used to online diagnose accidental faults using a vote algorithm integrated with a low-pass filter. We design a specific test bed to assess some core measures for fault diagnosis approaches, including the sensitivity, specificity precision, accuracy and F1-score. The test results show that the approach is robust, efficient and accurate.

1.4. Organization of article

This article is organized as follows: Section 2 introduces HMM modeling methods; Section 3 compares this proposal with the related works; Section 4 formulates the target problem; Section 5 details the accidental fault diagnosis approach based on HMMs; Section 6 experiments the approach by designing a specific test bed, and discusses the generalization of the approach; Section 7 gives some concluding remarks.

2. HMM modeling and analysis

An HMM is defined as a statistical model used to represent stochastic processes, where the states are not directly observed. When modeling a system, HMM separates the system into two conceptually independent paradigms: behavior and observation. Behavior refers to what the system really is; while observation to what the system exhibits. The observations are used for the recognition of the inner behavior. The initial state distribution π indicates the probabilities for the initial state of the system. The state transition matrix A controls the way the hidden state at time t is chosen given the hidden state at time $t-1$.

The emission matrix B connects the behavior between states and observations: if at a given time the behavior is known, how probably an observation sequence will occur. An HMM can be described as follows:

- N, number of states;
- M, number of observations;
- π, initial state probability distribution, $\sum_{i=1}^{N} \pi_i = 1$;
- A, transition matrix $\{a_{ij}\}$ for the probabilities from state s_i to state s_j,

$$\sum_{j=1}^{N} a_{ij} = 1, \quad 1 \leqslant i \leqslant N$$

- B, emission matrix $\{b_i(o_j)\}$ for the probabilities from states s_i to observations o_j,

$$\sum_{j=1}^{M} b_i(o_j) = 1, \quad 1 \leqslant i \leqslant N$$

Example 1 (HMM example). *An HMM used to simulate a system's health states and behaviors is illustrated by Figure 2. It defines two states (respectively Healthy and Faulty), and two observations representing that the output values respect (R) or violate (V) the functional constraints.*

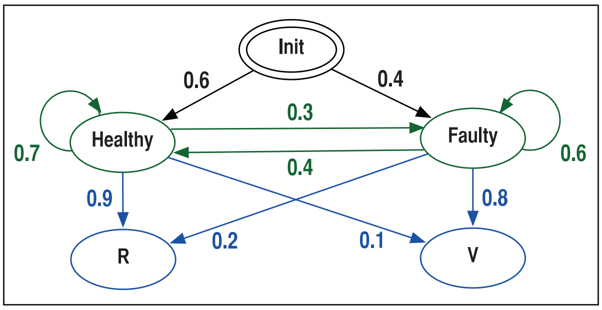

Figure 2. A two-state HMM.

The 3 matrices π, A and B are respectively: $\begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$, $\begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$ *and* $\begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$.

As shown in Figure 3, HMM, as an abstract model of a system, is statistically identical to the system's real behavior. The parameters in π, A and B can be obtained through a training process using the observation data and the Baum–Welch algorithm. Once all of these parameters are estimated, an HMM is able to compute, given an observation sequence, the maximum likelihood estimation of inner-state transition sequences using the Viterbi algorithm.[14] Thus, it can be used to estimate the system's current (past) state, predict the system's future state and compute the occurrence probability of a future observation sequence.

3. Related works

In Clarke and Zuliani,[15] the authors aimed to prove the validation of CPS's behavioral properties by statistical models. Our main ideas are quite similar: using statistical model to deduce the probability of verification results. They introduced a formal description in BLTL (bounded linear temporal logic) to interpret the probable result sequence, which is the same concept as the observation sequence in HMM. This work relied on the importance-sampling and cross-entropy (two classic statistical methods). It needed much more samples to give out a convincing conclusion. The reason behind is that BLTL treats each element in a result sequence in a pure statistical thus relatively independent manner, while the HMM modeling method ensures that the system components are dependent and thus the fault propagation is statistically preserved.

Some existing works[16–19] proposed HMM-based algorithms to diagnose or prognose system faults, where rich training data is required to estimate the parameters in HMMs. It is not applicable to the accidental faults due to its non-frequent characteristics. Smyth[18] and Ying et al.[19] aimed at fault detection and diagnosis based on HMMs under the single fault assumption.

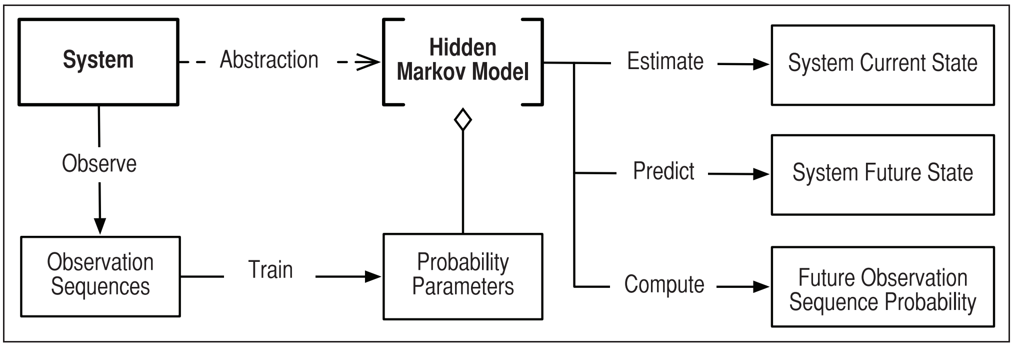

Figure 3. HMM training approach

Smyth[18] proposed to model the monitored system by linear difference equations, where the parameters were estimated in a training manner using the input and the observed output data of the system. They relied on the feedforward multi-layer neural network[20] and a simple maximum-likelihood Gaussian classifier[21] to produce the instantaneous estimates of the state probabilities, given the observations. Hence, fault detection occurred by observing changes in the values of estimated parameters. Finally, an HMM model, which combined the past state estimates and the current instantaneous estimates, was applied to determine the current state. The experimental results showed that the neural-Markov combination was significantly better than the combination of Gaussian–Markov in terms of elimination of false alarms. However, it is difficult to build a neural network for a large class of faults, while for the accidental fault diagnosis, the classes of fault must include all of the possibilities. This method is based on an important assumption of availability of labeled training data, which rules out applications where it is not possible to gather such data like the accidental faults.

Ying et al.[19] presented an HMM-based algorithm for fault diagnosis in systems with partial and imperfect tests. The initial state probabilities and state transition probabilities were defined using the failure rates of components. They constructed a fault dictionary via reachability analysis on directed graph models by denoting that a failure source was detected by a test or not. The emission probabilities were defined using this fault dictionary and the false alarm probabilities derived from tests using the central limit theorem.[22] All of these parameters were then adapted by the Baum–Welch algorithm. Once the parameters are estimated, the trained HMM is used to determine the current state by employing a sliding window Viterbi algorithm.

In our work, the modeling method for the states, initial and transition matrices is quite similar to the work of Yang et al.[19] The main difference consists of the observations and the parameter estimation for the emission matrix, which is the key problem for the HMM-based algorithms. As the faulty output caused by an accidental fault may not be continuous, it is possible for the system to behave normal after a sudden failure. Due to the non-frequent characteristics of accidental faults, its diagnosis in complex systems is restricted by the limited amount of faulty data. When the failure rates of hardware components are available, it is not reasonable to rely on the limited amount of faulty data to adapt the parameters in the initial and transition matrices using the Baum–Welch algorithm. We thus propose to fix these two matrices using the failure rate parameters of hardware components while estimate the emission probabilities. The estimation method is based on the principles of FTA and the maximum entropy in Bayesian probability theory. A fault propagation distribution is thus computed, whose parameters are adapted using the backward algorithm and observations. The parameterized HMMs are then used to online diagnose accidental faults using a vote algorithm integrated with a low-pass filter.

The method for diagnosing accidental faults using HMMs was first presented in our previous work.[23] Compared with our previous work,[23] this work has two important improvements: introducing a low-pass filter to the vote algorithm; and introducing a backward algorithm that adapts the emission matrix parameters to the state sequences derived from the observations. The experiments show that the fault diagnosis has been prominently improved.

4. Problem description

The accidental faults may occur on the hardware during system's execution. As their occurrence is infrequent, it is rather rare to have two or more hardware components being the source of accidental faults at exactly the same time. Therefore, it is reasonable to apply the following single-fault assumption to the target problem.

Assumption 1 (Single-fault assumption). *An occurrence of accidental fault denotes that at each time instant, only one hardware component is the fault source. Two components cannot be the source of accidental fault at the same time.*

The prior knowledge in this work consists of the failure rate parameters of hardware components and system specifications. The failure rate parameters are usually provided by the hardware manufacture according to the testing results before the hardware are supplied. They denote the fault occurrence probability either caused by design faults, physical wear and tear, environmental conditions or by a peculiar set of inputs/excitations given to the system. System specifications describe the expected functional constraints. They are used to online test the pass/fail of the output data of hardware components to build the observation sequences. The target problem is explained using the following example.

Example 2 (Problem description example). *During a finite execution time t, six time instants are observed for the system in Figure 4. Assume 5 functional constraints C_{fi} ($1 \leqslant i \leqslant 5$) are available in the system specification. After time t, the test results are obtained: R for respecting specification, V_I for single variable violating specification, and V_C for multiple variables violating functional constraint. The test results are shown in Table 1.*

When a wrong output from a component is observed, usually the outputs from other components may also be wrong due to the failure propagation. It is important to distinguish the output fault of a component and the source of fault. When a component is the fault source, many other components may behave wrongly (output is wrong). The

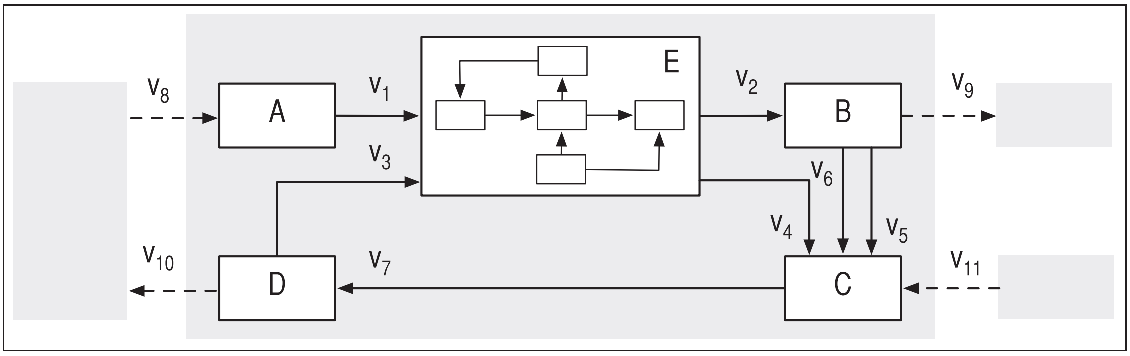

Figure 4. Problem description example.

Table 1. Output observations.

t	$C_{f1}(v_1,v_2,v_3)$	$C_{f2}(v_7,v_{11})$	$C_{f3}(v_9)$	$C_{f4}(v_4,v_5,v_6)$	$C_{f5}(v_2)$
1	R	R	R	R	R
2	R	R	R	R	R
3	V_C	R	R	V_C	R
4	R	R	R	R	V_I
5	V_C	R	R	V_C	V_I

modeling method thus should consider the dependencies between the hardware elements and the fault propagation between them.

The accidental fault diagnosis approach should allow online analysis, which implies that the method needs to be computation-economic and easy to be implemented by most onboard embedded systems.

5. Online diagnosis of accidental fault

5.1. Overview of proposed approach

The modeling of HMM follows two steps: first, determine the structure of HMM, i.e. the states and observations; then, estimate the parameters for the three matrices.

An overview of the proposed approach is illustrated by Figure 5. For a monitored system that contains n components, $n + 1$ hidden states are modeled. Each state represents, at time t, which component has encountered an accidental fault. The observation sequences are derived from the test results with respect to the functional constraints. The parameters in the initial state distribution and the state transition matrix are computed using the failure rates of hardware components. The emission matrix parameters are estimated using a failure propagation algorithm based on the principles of FTA and the maximum entropy in Bayesian probability theory. Relying on the Viterbi algorithm, the state sequences are derived from the observation sequences. They are then used to diagnose the accidental faults using a vote algorithm through a low-pass filter and to revise the parameters in the emission matrix through a backward algorithm.

Compared with our previous work,[23] this work introduces the vote algorithm through the low-pass filter and the backward algorithm for revising the failure propagation distribution to improve the diagnosis results.

5.2. System states

System states represent that at time t, which component is the source of accidental fault. A system with n hardware components is thus model by $n + 1$ states $\{s_i\}$ $(0 \leqslant i \leqslant n)$. Here s_0 represents that none of components encounters accidental faults. Here s_i represents that the ith component is the fault source.

5.3. Observations

Definition 1 (Physical variable). *A component C has n_I inputs and n_O outputs variables, which connect C with other components. These variables are defined as physical variables of C.*

During the execution of a monitored system, the test results of functional constraints are available. The observations represent that physical variables respect or violate the functional constraints. If m physical variables v_j $(1 \leqslant j \leqslant m)$ are involved, m HMMs $\{h_j\}$ are modeled for this system. Here h_j describes how a variable v_j behaves in terms of functional constraints during system's execution.

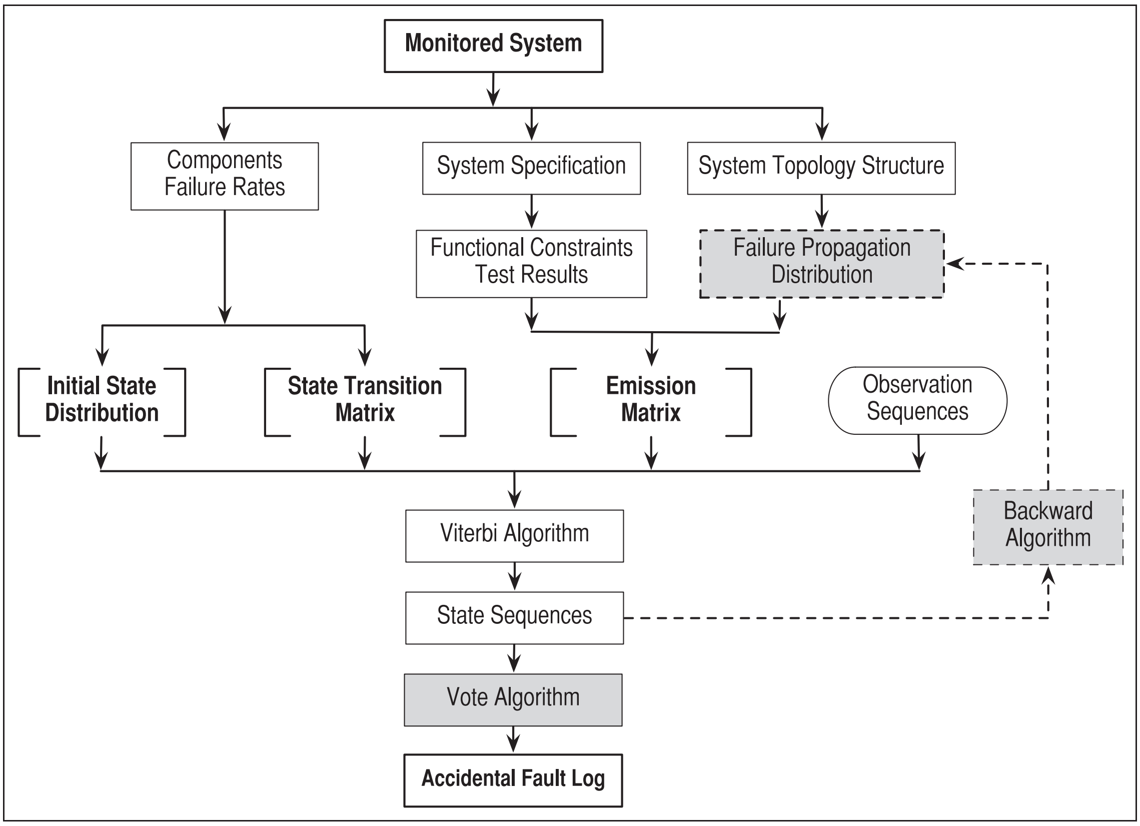

Figure 5. Approach overview.

It is possible to generate three kinds of observations for v_j: respecting, independent violating or coupling violating functional constraints.

Definition 2 (Respecting (R)). *When a physical variable does not violate any functional constraint, this observation is defined as Respecting.*

Definition 3 (Independent violation (V_I)). *When a physical variable v violates a functional constraint that only contains v, i.e.* $C(v)$, *this observation is defined as independent violation.*

Definition 4 (Coupling Violation (V_C)). *When v violates a functional constraint containing v as well as other variables, i.e.* $C(v, v_1, v_2, \ldots)$, *this observation is defined as coupling violation.*

Therefore, each HMM h_j has these three kinds of observations: R, V_I and V_C. The structure of HMMs that model the monitored system is thus defined, as shown in Figure 6.

An intuitive way for modeling the observations is to directly use two observed statuses: respected (R) and violated (V). In this work, we provide more details in the observation by distinguishing V_I and V_C. The observations will be used to decide the probabilities in the emission matrix, which depends on the topology structure connecting by physical variables. We rely on the principles of FTA to compute the influence weight of variables, which requires as much as detailed test results about each variable.

5.4. Initial probability distribution π

For a system with n components, the failure rate parameters $\{\omega_i\}$ $(1 \leqslant i \leqslant n)$ of hardware components $\{C_i\}$ are usually provided by its manufacture or determined by the MTBF (mean time between failure). Without prior knowledge, $\{\omega_i\}$ are determined by $\sum_{i,j=1}^{n} \left|\omega_i - \omega_j\right| = 0$. According to the definition of HMM, $\pi = \{\pi_i\}$ are directly computed as follows:

$$\pi_0 = \frac{\prod_{i=1}^{n} (1 - \omega_i)}{\prod_{i=1}^{n} (1 - \omega_i) + \sum_{i=1}^{n} \omega_i} \tag{1}$$

$$\pi_i = \frac{\omega_i}{\prod_{i=1}^{n} (1 - \omega_i) + \sum_{i=1}^{n} \omega_i}, 1 \leqslant i \leqslant n \tag{2}$$

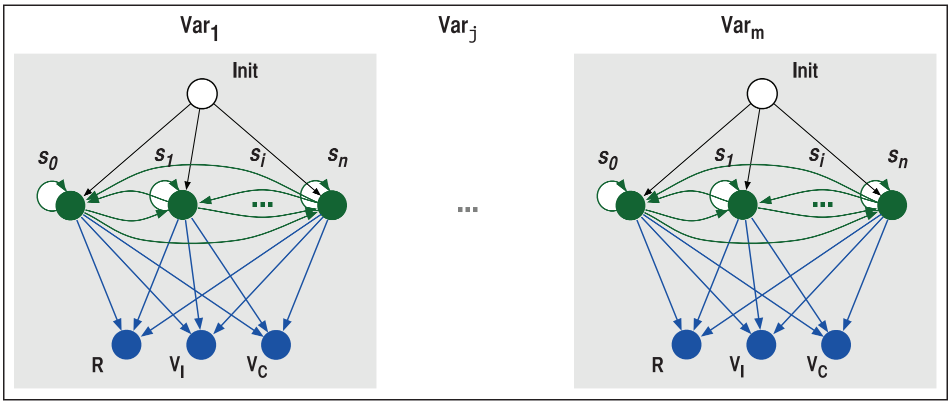

Figure 6. HMMs for *m* observation variables.

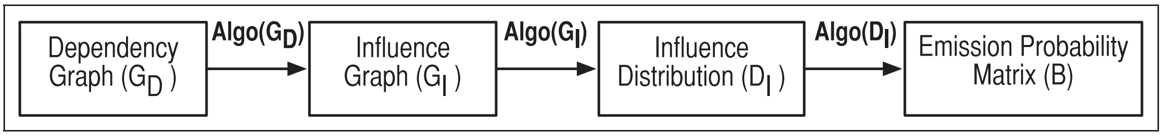
Dependency Graph (G$_D$)
Algo(G$_D$)
Influence Graph (G$_I$)
Algo(G$_I$)
Influence Distribution (D$_I$)
Algo(D$_I$)
Emission Probability Matrix (B)

Figure 7. Emission matrix algorithms.

5.5. *Transition probability matrix A*

Here $\{a_{ij}\}$ ($0 \leqslant i, j \leqslant n$) represents the probabilities of the transition from state i to state j. It is computed using the failure rate parameters $\{\omega_i\}$. When the system transits from current state to the no fault state s_0, the transition probability is $\prod_{k=1}^{n}(1-\omega_k)$. When the system transits to a faulty component state s_j ($j \neq 0$), the probability is $\omega_j \cdot \prod_{k=1,k\neq j}^{n}(1-\omega_k)$. After normalization, A is determined as follows:

$$a_{i0} = \frac{\prod\limits_{k=1}^{n}(1-\omega_k)}{\prod\limits_{k=1}^{n}(1-\omega_k) + \sum\limits_{n=1}^{n}\left(\omega_n \prod\limits_{k=1,k\neq j}^{N}(1-\omega_k)\right)}, \quad 0 \leqslant i \leqslant n \tag{3}$$

$$a_{ij} = \frac{\omega_j \prod\limits_{k=1,k\neq j}^{n}(1-\omega_k)}{\prod\limits_{k=1}^{n}(1-\omega_k) + \sum\limits_{p=1}^{n}\left(\omega_p \prod\limits_{k=1,k\neq j}^{n}(1-\omega_k)\right)}, \quad 0 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant n \tag{4}$$

5.6. *Emission probability matrix B*

Here $\{b_{ij}(O)\}$ ($0 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, $O \in \{R, V_I, V_C\}$) represents, if a component encounters accidental faults, the probability that it will influence the observations. More precisely, how probably the physical variables will violate the functional constraints. Inspired by the principles of FTA and the principle of maximum entropy in Bayesian probability theory, we propose the algorithms for computing B, as shown in Figure 7.

5.6.1. Computing influence graph G_I.

Definition 5 (Dependency graph G_D). *A dependency graph represents the dependency between system's components. It is directly derived from the system's topology structure.*

Definition 6 (Influence graph G_I). *An influence graph is a graph representing how components' accidental fault influences its output physical variables. Influence graph is topologically identical to the dependency graph, with supplementary influence weights indicating the probability that the component C_i influences its output variables $\{v_j\}$. An influence layout algorithm is proposed to compute the influence weights $G_I(i, j)$.*

Example 3 (Influence graph example). *Figure 8 is an influence graph with 9 components and 15 physical variables. Based on the dependency graph, the influence weights $G_I(1, 2)$ and $G_I(2, 3)$ respectively indicate whether component C_1 (respectively C_2) is the fault source, the probability that v_2 (respectively v_3) will be influenced to produce a wrong output.*

Algorithm 1 (Influence layout algorithm). *An influence graph with n components and m variables can be solved by applying linear programming to Equations* (5)–(8).

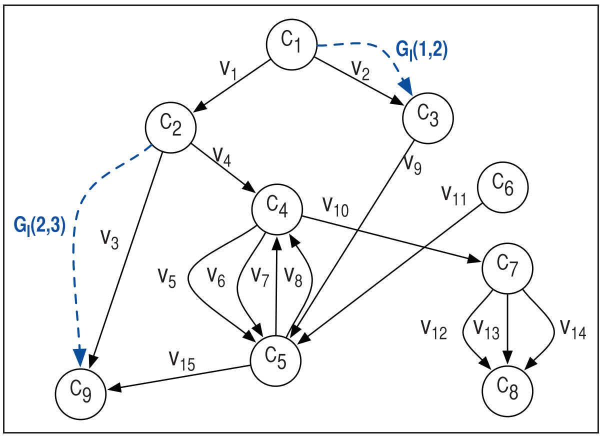

Figure 8. Influence graph example.

Equation (5) *normalizes all of the influence weights*:

$$\sum_{i=1}^{m} v_i = 1 \tag{5}$$

Equation (6) *ensures that a component must have some impact on its output variables*:

$$v_i > 0 \tag{6}$$

Equation (7) *represents the failure effect caused by a component C (denoted as* ε(*t*)*), which is equal to the difference between the sum of C's output weights and the sum of its input weights*:

$$\sum_{in} v_i - \sum_{out} v_j = \varepsilon(t) \tag{7}$$

Here ε(t) *varies along with the execution time t, and is measured using the state sequences and the backward algorithm (Algorithm 6, explained later in Section 5.9).*

The maximum entropy principle states that the probability distribution which best represents the current state of knowledge is the one with largest entropy. Equation (8)*ensures that the difference between variables must be minimized according to this principle*:

$$\min_{v_1 \ldots v_m} \sum_{i=1}^{m} \left| v_i - \frac{1}{m} \sum_{j=1}^{m} v_j \right| \tag{8}$$

5.6.2. Computing influence distribution D_I. The influence graph gives out the influence weight from a component to its direct output variables. This faulty impact will propagate through the dependency structure of the system. The propagated weights are recorded in the influence distribution.

Definition 7 (Influence distribution D_I). *When component C_i is the fault source, the failure influence it causes to all of the physical variables $\{v_j\}$ is defined as $D_I\,(i, j)$.*

Ideally, D_I can be derived from exhaustive simulation or tests. If enough simulation or test scenarios that cover all of the variable ranges are available, the parameters in D_I can be almost the same as the real values. The problem is that, for the non-frequent accidental fault, the time for obtaining enough training data by exhaustive simulation (test) is not definitive. To make a compromise between the computation time and the parameter precision, we compute D_I using the *influence graph*. The algorithm is based on the principle of FTA, a top-down deductive failure analysis method widely used in the FDI techniques, in which an undesirable state of a system is analyzed using Boolean logic to combine a series of lower-level events. It is an heuristic algorithm, in which the coefficients will be precise with respect to the problem. In this work, we propose the following influence distribution algorithm to compute the coefficients of fault propagation.

Algorithm 2 (Influence distribution algorithm). *When the output variables of component C_i violates some functional constraints, it is either because C_i has encountered an accidental fault, or because one of its dependent components C_j has encountered an accidental fault and thus C_i is influenced due to the failure propagation.*

- *If C_i has encountered an accidental fault, it will probably generate an output violation. The reason why it is not a definitive violation is that the coupling variables referred to by the functional constraints may also be influenced. This probability is defined as D_A.*
- *If C_i depends on C_j (C_i directly or indirectly takes C_j's output as input), and an accidental fault occurs on C_j, C_i will probably have an output violation. It is not a definitive violation, because either the coupling variables referred to by the functional constraints may also be influenced; or the intermediate component's design has fault-tolerance consideration. This probability is defined as D_B.*

As D_A and D_B are independent. If v_j directly depends on c_i, at the same time, it can indirectly depend on c_i due to the control feedback loop. Therefore, we have

$$D_I(i,j) = D_A(i,j) + D_B(i,j) \tag{9}$$

- *If variable v_j does not depend on component C_i (C_i has no path to v_j),*

$$D_I(i,j) = 0 \tag{10}$$

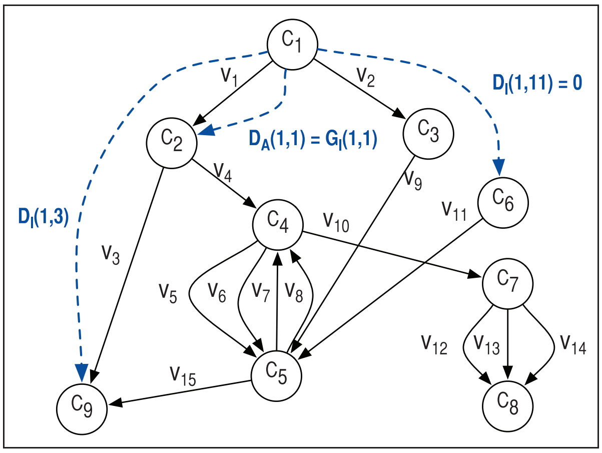

Figure 9. Influence distribution example.

- *If v_j directly depends on C_i,*

$$D_A(i,j) = G_I(i,j) \tag{11}$$

- *If v_j indirectly depends on C_i, let $\{v_k\}$ $(k \neq j)$ be the output variables of C_i, and C_k be the target component of v_k. The $D_B(i, j)$ can be recursively computed as follows*:

$$D_B(i,j) = \sum_k G_I(i,k)_I(k,j) \tag{12}$$

Example 4 (Influence distribution algorithm example). *An example of influence distribution algorithm is given by Figure 9. We enumerate three cases.*

1. $D_I(1, 11)$: *As there is no path from C_1 to v_{11}, v_{11} is independent from C_1. The influence weight from C_1* to v_{11} is $D_I(1,11) = 0$.
2. $D_I(1, 1)$: *As v_1 is the direct output from C_1 and there is no control feedback loop, the influence weight from C_1 to v_1 is* $D_I(1,1) = D_A(1,1) = G_I(1,1)$.
3. $D_I(1,3)$: *As v_3 indirectly depends on C_1, the influence weight from C_1 to $v_3(D_I(1,3))$ is computed using Equation* (12), *as follows*:

$$\begin{aligned} D_I(1,3) = G_I(1,1) \cdot D_I(2,3) + G_I(1,2) \\ \cdot D_I(3,3) = G_I(1,1) \cdot D_A(2,3) \\ + 0 = G_I(1,1) \cdot G_I(2,3) \end{aligned}$$

5.6.3. Emission probability matrix B. **Algorithm 3** (Emission matrix algorithm). *For the HMM h_j of variable v_j, we use the following algorithm to compute the probability for the observations R, V_I and V_C from state s_i.*

- *For the independent violation V_I, its probability is directly deduced from D_I.*

$$b_{ij}(V_I) = D_I(i,j) \tag{13}$$

- *For the coupling violation V_C, if v_j violates functional constrains containing other variables $\{v_k\}$ $(k \neq j)$, assume v_k appears n_k times in all of the functional constraints, and v_j appears n_j times, the coupling violation probability is computed as follows*:

$$b_{ij}(V_C) = \frac{n_j \cdot D_I(i,j)}{\sum_k n_k \cdot D_I(i,k)} \tag{14}$$

- *For the non-violation observation R, its probability is*

$$b_{ij}(R) = 1 - b_{ij}(V_I) - b_{ij}(V_C) \tag{15}$$

Obviously, the healthy state (s_0) will lead to $b_{0j}(R) = 1$, $b_{0j}(V_I) = 0$ *and* $b_{0j}(V_C) = 0$.

5.7. Viterbi algorithm

Once the parameters in HMMs are estimated, it can be used to decode an observation sequence $\mathcal{O} = \{o_t\}$ into the most likely state sequence $\mathcal{Q} = \{q_t\}$. This is computed using the Viterbi algorithm, a dynamic programming algorithm for finding the most likely sequence of hidden states (called a Viterbi path).

Suppose that T time instants are observed for a system with m physical variables and n components. Therefore, m HMMs with $n + 1$ states are modeled, denoted as h_j, $1 \leqslant j \leqslant m$. For each $\{h_j\}$, an observation sequence of length T is thus obtained.

To find the single optimal state sequence, the Viterbi algorithm defines a variable $\delta(i)$ in

$$\delta_t(i) = \max_{q_1 q_2 \ldots q_{t-1}} P\{q_1 q_2 \ldots q_t = s_i, o_1 o_2 \ldots o_t | h_j\} \tag{16}$$

which represents the maximum probability along a single path that accounts for the first t observations and ends at state s_i.

By induction, $\delta_{t+1}(k)$ can be computed as

$$\delta_{t+1}(k) = \max_i [\delta_t(i) a_{ik}] b_k(o_{t+1}) \tag{17}$$

To retrieve the state sequence, we need to keep the track of the state that maximizes $\delta_t(i)$ at each time t, which is done by constructing the following array, where $\varphi_{t+1}(k)$ is the state at time t from which a transition to state s_j maximizes the probability $\delta_{t+1}(k)$

$$\varphi_{t+1}(k) = \operatorname*{argmax}_{1 \leqslant i \leqslant n+1} [\delta_t(i) a_{ik}] \tag{18}$$

The Viterbi algorithm for finding the optimal state sequence becomes as follows.

- Initialization:

$$\delta_1(i) = \pi_i b_i(o_1), 1 \leqslant i \leqslant N \tag{19}$$

$$\varphi_1(i) = 0(\text{no previous states}) \tag{20}$$

- Recursion:

$$\delta_t(j) = \max_{1 \leqslant i \leqslant n+1} [\delta_{t-1}(i)a_{ik}]b_k(o_t), \quad 2 \leqslant t \leqslant T,\ 1 \leqslant k \leqslant n+1 \tag{21}$$

$$\varphi_t(k) = \operatorname*{argmax}_{1 \leqslant k \leqslant n+1} [\delta_{t-1}(i)a_{ik}], \quad 2 \leqslant t \leqslant T,\ 1 \leqslant k \leqslant n+1 \tag{22}$$

- Termination:

$$P = \max_{1 \leqslant i \leqslant n+1} [\delta_T(i)] \tag{23}$$

$$q_T = \operatorname*{argmax}_{1 \leqslant i \leqslant n+1} [\delta_T(i)] \tag{24}$$

- The optimal state sequence can be retrieved by backtracking

$$q_t = \varphi_{t+1}(q_{t+1}), \quad t = T-1, T-2, \ldots, 1 \tag{25}$$

The computation complexity of Viterbi algorithm is $O(K^2L)$, where K and L are respectively the number of HMM states and the length of $\mathcal{O}$. To online diagnose accidental faults for time t, we use the observation sequences in the past time of t according to the feature of Markov process. When t is too long, we might have huge quantity of observation sequences, which makes the computation time increases, a sampling window of w is thus introduced to ensure the efficiency of the online diagnosis.

5.8. Vote algorithm

At time t, each HMM h_j obtains an optimal state sequence $\mathcal{Q} = \{q_t\}$ using the Viterbi algorithm. As the system is modeled by m HMMs, a state sequence matrix of size m by t is built using the m optimal state sequences, denoted as $\mathcal{Q}_j = \{q_t(j)\}$. Here $\mathcal{Q}_j$ is used to diagnose accidental faults using a vote algorithm and also provide feedback by a backward algorithm to revise the emission matrix parameters.

Algorithm 4 (Vote algorithm). *For time t, each HMM computes the most probable fault source respectively using its state sequence* $\mathcal{Q} = \{q_t\}$, *and vote to synthesize a globally optimized result. In order to avoid hash synthesis, it would be a good idea to wait for several periods to see if the vote is stable. More precisely, to diagnose the potential fault source at time t, it needs to observe from instant* $t - w$ *to t to see whether the trend is stable, where w is the time window. This can be considered as a low-pass filter for the vote result. The larger this time window is, the fewer false alarms will be given by the method. In practice, this value decides the trade-off between the timeliness of online analysis and cost of false alarm handling. Therefore, a new state sequence* $\widehat{\mathcal{Q}} = \{\hat{q}_t\}$ *is derived thought the low-pass filter (given by Algorithm 5).*

At time T, the optimal state sequence matrix after using the low-pass filter is $\widehat{\mathcal{Q}_j} = \{\hat{q}_t(j)\}$, $1 \leqslant j \leqslant m$ *and* $T - w + 1 \leqslant t \leqslant T$.

The global vote results are represented by a vote matrix $\mathcal{V} = \{v_t(i)\}$, $0 \leqslant i \leqslant n$. *It computes, at time t, the appearance times of the state* s_i *among the m state sequences.*

$$v_t(i) = \sum_{j=1}^{m} \varphi(\widehat{q_t}(j)) \tag{26}$$

Algorithm 5. (Low-pass filter algorithm).

```
public static int[] filter(int[] original, int delayWindow,
int HEALTH_INDEX) {
      int[] newSeq = new int[original.length];
      for (int i = 0; i < delayWindow; i++)
        newSeq [i] = original[i];
      for (int i = delayWindow; i < original.length; i++) {
        int v = original[i];
        int endIndex = -1;
        boolean match = true;
        for (int j = i; j > i - delayWindow; j–) {
          if (original[j] != HEALTH_INDEX
          && original[j] != v) {
            match = false;
            break;
          }
          if (original[j] == v)
            endIndex = j;
        }
        if (!match)
          newSeq[i] = original[i];
        else {
          if (endIndex == -1)
            newSeq[i] = original[i];
          else {
            // smooth
            for (int j = endIndex; j < = i; j++)
              newSeq[j] = v;
          }
        }
      }
      return newSeq;
}
```

Table 2. State sequence matrix.

t	1	2	3	4	5
$\mathcal{Q}_1$	s_0	s_2	s_1	s_1	s_1
$\mathcal{Q}_2$	s_0	s_1	s_0	s_0	s_1
$\mathcal{Q}_3$	s_0	s_1	s_3	s_2	s_2

Table 3. Filtered state sequence matrix.

t	1	2	3	4	5
$\mathcal{Q}_1$	s_0	s_2	s_1	s_1	s_1
$\mathcal{Q}_2$	s_0	s_1	s_1	s_1	s_1
$\mathcal{Q}_3$	s_0	s_1	s_3	s_2	s_2

$$\varphi(\hat{q}_t(j)) = \begin{cases} 1 & \widehat{q}_t(j) = s_i \\ 0 & \widehat{q}_t(j) \neq s_i \end{cases} \tag{27}$$

In order to observe the trends of fault occurrence along with the time, we introduce a variation matrix to measure the fault trends between two neighboring time instants. This matrix is denoted as $\mathcal{U} = \{u_t(i)\}$:

$$u_t(i) = \begin{cases} 0 & t = T - w + 1 \\ v_t(i) - v_{t-1}(i) & T - w \leqslant t \leqslant T \end{cases} \tag{28}$$

The accidental fault log $\mathcal{F} = \{f_t\}$ is then derived from the matrices $\mathcal{U}$ and $\mathcal{V}$:

$$f_t = \begin{cases} \{s_i | \max u_t(i)\} & card(f_t) = 1 \\ \{s_i | s_i \in \max u_t(i) \wedge \max v_t(i)\} & card(f_t) \geqslant 2 \end{cases} \tag{29}$$

Example 5 (Vote algorithm example). *For a time duration 5, the state sequences for 3 variables derived from the Viterbi algorithm are shown in Table 2. The system has 3 components, and the time window is 5 time units. The state sequence matrix after low-pass filtering is Table 3. The vote matrix derived from state sequence matrix is given by Table 4, and the variation matrix is given by Table 5. By applying the vote algorithm, the accidental fault log $\mathcal{F}$ is shown in Table 6.*

5.9. Backward algorithm

This backward algorithm adapts the emission matrix parameters to the state sequences derived from the observations. It is used to revise the parameter $\varepsilon(t)$ in Equation (7) when computing the influence distribution. The $\varepsilon(t)$ represents the fault impact that a component causes to its output variables.

Table 4. Vote matrix.

t	1	2	3	4	5
s_0	3	0	0	0	0
s_1	0	2	2	2	2
s_2	0	1	0	1	1
s_3	0	0	1	0	0

Table 5. Variation matrix.

t	1	2	3	4	5
s_0	0	−3	0	0	0
s_1	0	2	0	0	0
s_2	0	1	−1	1	0
s_3	0	0	1	−1	0

Table 6. Accidental fault log.

t	1	2	3	4	5
$\mathcal{F}$	s_0	s_1	s_3	s_2	s_1

Algorithm 6 (Backward algorithm). *At the initial time, none of components is faulty, thus the fault impact of the component C_i is 0. If $t \geqslant 1$, the value of $\varepsilon_i(t)$ is computed using the vote matrix $\mathcal{V}$ obtained from the state sequence matrix $\mathcal{Q}$:*

$$\varepsilon_i(t) = \begin{cases} \dfrac{\sum_t v_t(i)}{(n+1)\cdot t} & 1 \leqslant t < w \\ \dfrac{\sum_{t-w+1}^{t} v_t(i)}{(n+1)\cdot w} & t \geqslant w \end{cases} \tag{30}$$

6. Experimental results

It is difficult to evaluate a fault diagnosis method for accidental faults, because it is too costly to monitor accidental fault data from real systems. We thus design a specific test bed to randomly generate a large amount of test cases to assess some measures for fault diagnosis approaches: the sensitivity, specificity precision, accuracy and F1-score.

6.1. Experiment measures

To give the experiment measures for our fault diagnosis approach, the contingency matrix is given in Table 7. The system is either healthy (s_0) or faulty (s_i, $i \neq 0$). The diagnosis state derived from the HMM approach is either healthy (s_0) or faulty as same as the system state (s_i), or faulty but different from the system state (s_j, $j \neq i$). Therefore, the test results can be *hit faulty state, hit healthy state, false alarm, error alarm*, or *miss healthy state*.

Table 7. Contingency matrix.

		System state	
		Faulty s_i	Healthy s_0
HMM state	Faulty s_i	Hit faulty state s_i ($\mathbf{h}_i$)	False alarm (**f**)
	Faulty s_j	Error (**e**)	
	Healthy s_0	Miss healthy state (**m**)	Hit healthy state ($\mathbf{h}_0$)

In pattern recognition and information retrieval with binary classification, the following measures are usually used to assess the performance of the statistical approaches.[24]

- *Sensitivity (recall rate)* relates to the ability to identify an accidental fault correctly:

$$\text{Sensitivity} = \frac{\sum h_i}{\sum h_i + \sum e + \sum m} \tag{31}$$

The *miss rate* can be derived from it:

$$\text{Miss rate} = 1 - \textit{sensitivity} \tag{32}$$

- *Specificity* relates to the ability to exclude an accidental fault correctly:

$$\text{Specificity} = \frac{\sum h_0}{\sum h_0 + \sum f} \tag{33}$$

- *Precision* relates to the ability to avoid the false alarm:

$$\text{Precision} = \frac{\sum h_i}{\sum h_i + \sum e + \sum f} \tag{34}$$

The *false alarm rate* can be derived from it:

$$\text{False alarm rate} = 1 - \textit{precision} \tag{35}$$

- *Accuracy (hit rate)* relates to the ability to diagnose system's state correctly:

$$\text{Accuracy} = \frac{\sum h_i + \sum h_0}{\sum h_i + \sum h_0 + \sum e + \sum m + \sum f} \tag{36}$$

- *F1-score* is a measure of a statistic test's accuracy.

$$\text{F1} - \text{score} = 2 \times \frac{\textit{precision} \times \textit{recall}}{\textit{precision} + \textit{recall}} \tag{37}$$

6.2. Test bed

The pattern of system instance generated by the test bed follows the classical definition: a system has several components; components are connected by physical variables. A variable can be read by multiple components, but can only be written by one component. System itself has input and output, which is an encapsulation of some component's input/output variable according to system architecture.

Component, when encountering an accidental fault, will output, in a high probability p_1 (0.95), some non-functional-compliant data. A normal component, when dealing with a faulty input, tends to output also an faulty output in probability p_2 (0.6). Since component may have inner states, this negative impact may last for some time (less than p_3 (10 time units)), with a probability p_4 (0.5). Due to the incompleteness of functional constraints, an actual fault will only be identifiable in a probability p_5 (0.8) by applying these checks.

The five parameters above $p_1 - p_5$ vary from system to system. Some of them cannot even be easily represented by probability, for example, a good fault tolerance design will make p_2 and p_4 very specific to compute. In this article, in order to maximize the generality, all these parameters are excluded for the system configuration diversity. They are defined by constants. These constants are intent to cover as large as possible the real system pattern space.

A random test configuration is composed of the following parameters.

- *Component number*: defines the total number of component in this system instance.
- *Min/Max component failure rate*: each component will be distributed a failure rate falling into this range. Higher the rate, more likely to encounter an accidental fault.
- *Min/Max component output/input variable number*: each component will have a number of input/output variables, and their quantity is limited by the range respectively.
- *Min/Max accidental fault number*: during a simulation, defines how many accidental faults will be generated.
- *Simulation time*: defines the observation time for each system simulation.
- *Time window*: defines the sampling window for observations.

6.3. *Experimental results*

We assess the performance of the proposed approach through the measures including sensitivity (recall rate), specificity, precision, accuracy and F1-score. We have generated more than 1.4 × 10^5 test cases by varying the parameters in Table 7. The objective is to analyze the variation trends of there measures along with the system's scale, the sparsity of accidental faults and the sampling window size. In addition, as a low-pass filter algorithm is introduced to compute the optimal state sequence, we respectively analyze the measure values for the methods without and with low-pass filter to assess the effectiveness of the low-pass filter algorithm.

Table 8. Parameters in test cases.

Parameter	Value
Component number	10–90
Failure rate of hardware component	5×10^{-4}–7×10^{-4}
Output variable number per component	2–5
Accidental fault number	1–26
Simulation time units	1000–2000
Sampling window size	3–30

6.3.1. Experimental results by average component number.

Figure 10 gives the average measure values varying with the average component number for both methods. The following conclusions are derived from the test results.

- The specificity and accuracy are nearly 100% in both methods. This phenomenon is quite obvious. For the non-frequent accidental faults, its occurrence times (limited from 1 to 26 in our experiments) during the simulation (from 1000 to 2000 time units) is far less the number of normal states in the system. This explains why the average values

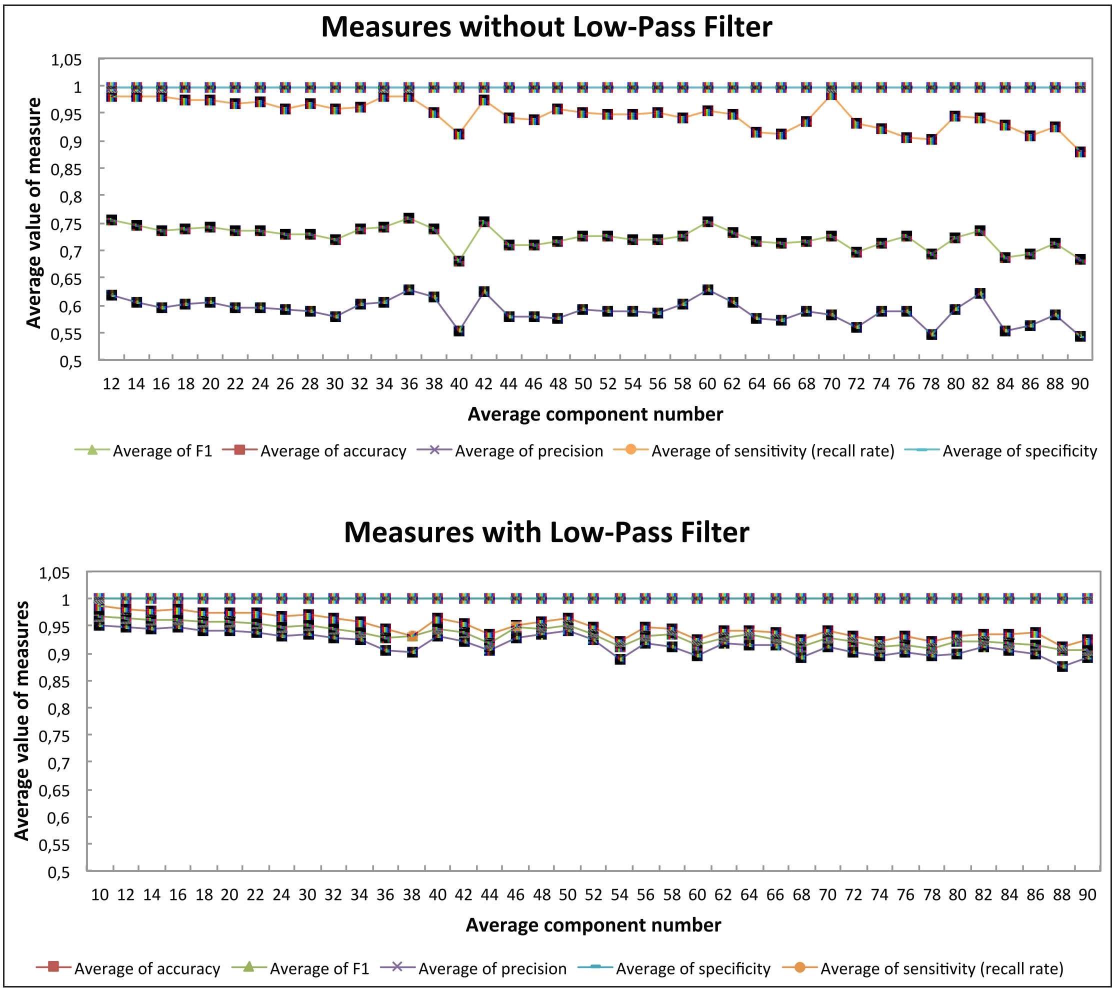

Figure 10. Measures with average component number from 10 to 90.

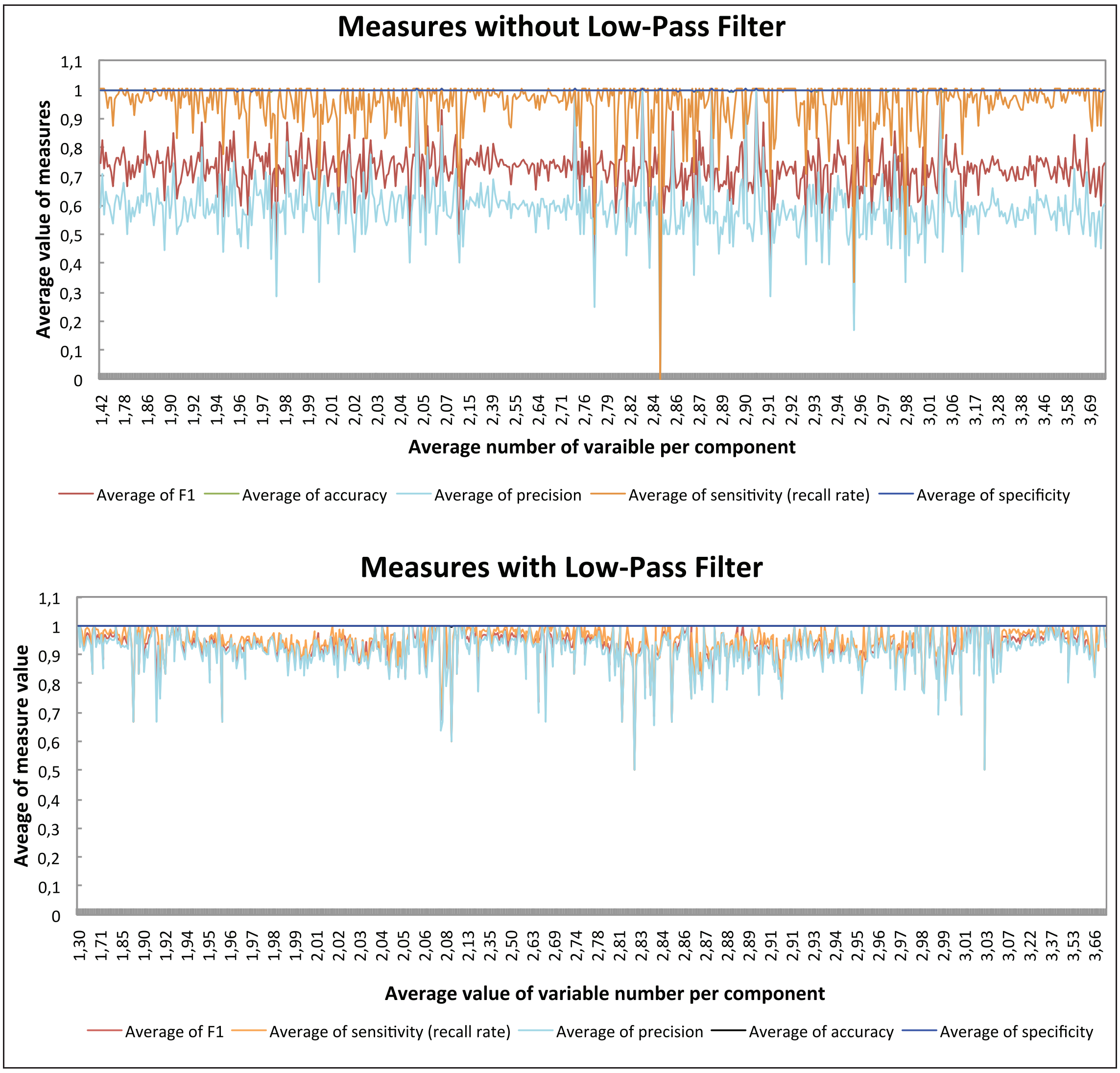

Figure 11. Measures with average output variable number per component from 1.5 to 4.

of specificity and accuracy are almost 100% in all of the following test results.

- The measures are largely improved by the low-pass filter.
- The method is robust with respect to the number of components (the scale of system).

6.3.2. Experimental results by average output variable number. Figure 11 in page 19 gives the average measure values varying with the average number of output variables per component for both methods. The following conclusions are derived from the test results.

- The measures are largely improved by the low-pass filter.
- The method is robust with respect to the average number of variables per component (the scale of system).

6.3.3. Experimental results by average accidental fault number. Figure 12 in page 20 gives the average measure values varying with the average accidental fault number during the simulation. The following conclusions are derived from these test.

- The measures are largely improved by the low-pass filter.
- The method is robust with respect to the number of accidental faults (the sparsity of data).

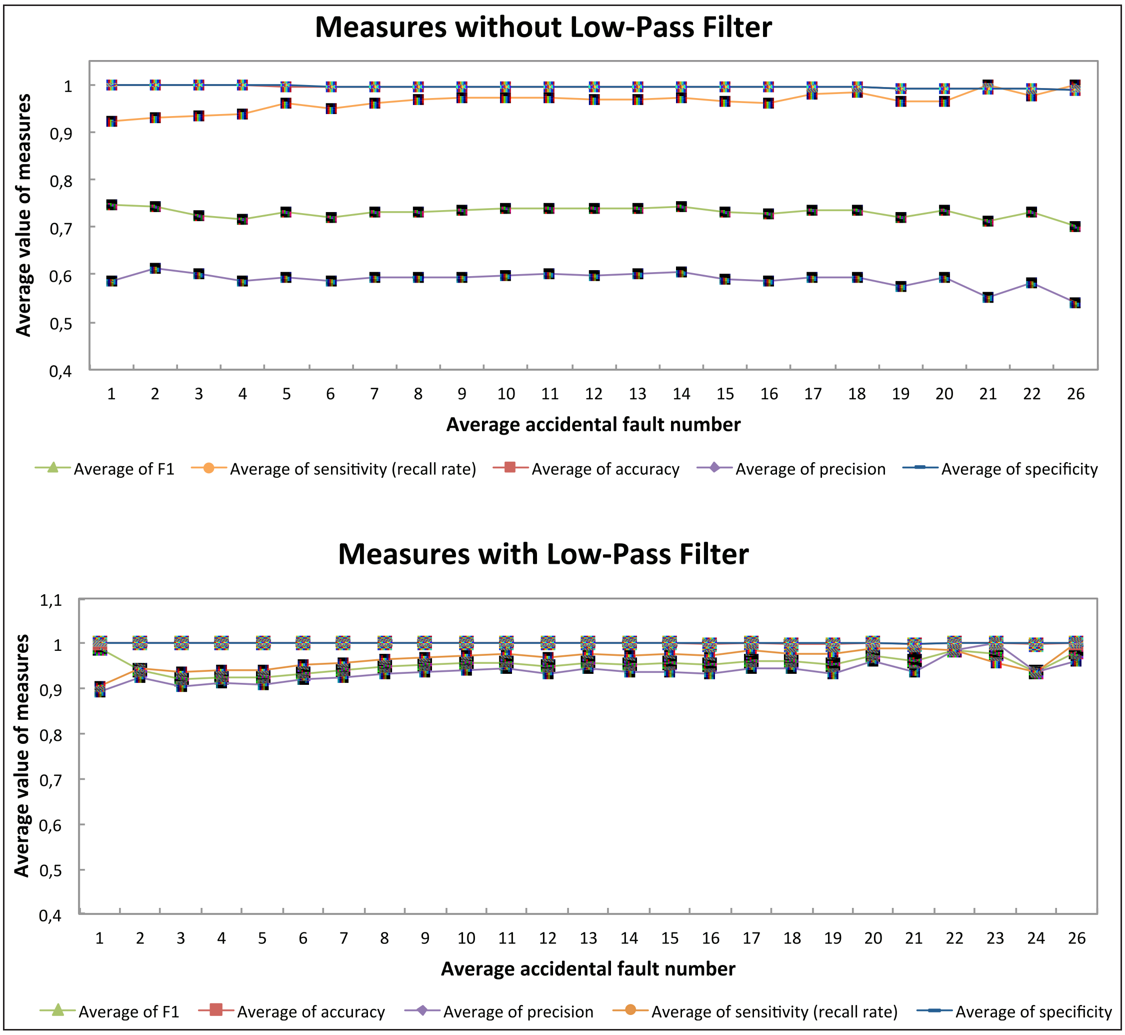

Figure 12. Measures with average accidental fault number from 1 to 26.

6.3.4. Experimental results by the size of sampling window. Figure 13 gives the average measure values varying with the size of the sampling window. When the time window is larger than 3 time units, the measures become stable and are not effectively impacted by the size of time window. This ensures the efficiency of online analysis.

6.4. Discussion: multiple fault diagnosis

Our base hypothesis in this work is that at any moment, at most only one component can be the source of accidental failure (which indeed implies that the old fault will disappear when a new one occurs). It is true that when there is only one fault source in a system, many hardware components may behave wrongly. As the Assumption 1 defined the states in the HMM that represent the component of fault source. In other words, a component that is not the fault source may be faulty (the outputs violate the functional constraints) due to the fault propagation. The modeling method thus has considered the dependencies between the hardware elements and the fault propagation between them.

We fully understand that at first glance, the single fault source assumption makes the approach not applicable for real-world scenarios because all can go wrong at any moment. However, it should be known that this simplification is only a trade-off for online fault analysis performance: our method can deal with the case that ‘‘at any moment t, at most K components of failure source can simultaneously occur ($K \geqslant 1$)’’. Therefore, 2^K states will be modeled in one HMM. The only issue is that, for an HMM with 2^K states, the analysis time will be quite long, which could degrade the timeliness of online fault analysis.

To be more precise, in our original work, we do handle the problem in a general way: inner states are the combination of the fault source component, the emission matrices are the overlap (normalized accumulation) of related

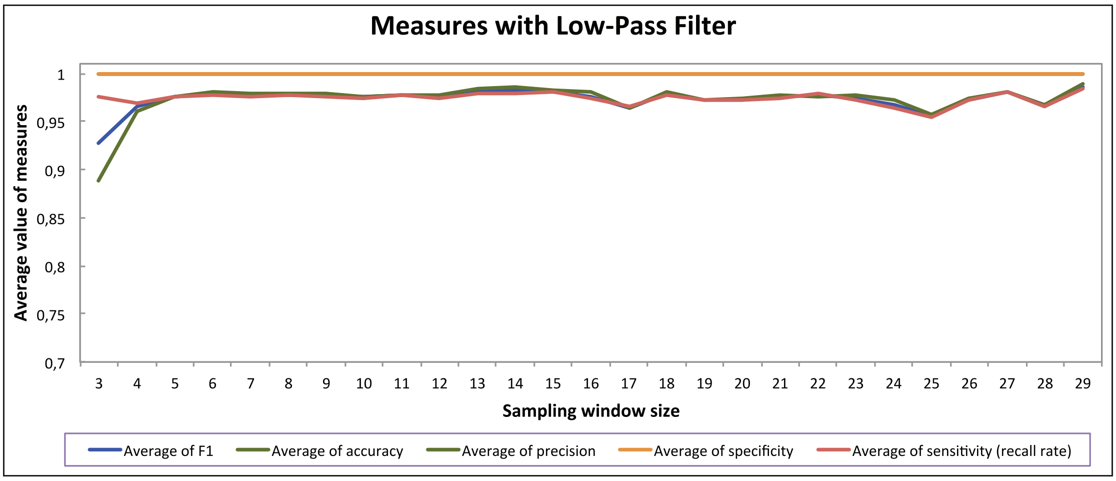

Figure 13. Measures with sample window size from 3 to 30.

states' influence distribution, etc. The test result is also quite positive (average measures are also $>$ 90%). The problem we encountered is that from $K > 4$, for a system having more than 30 components, the analyzing time (on a PC laptop) will exceed the cycle period of a typical embedded system (e.g. 50 ms), which compromises ''online analysis''. Therefore, we found that this method, although it had high accuracy, required a computing trade-off for large-scale systems. That is why finally we decide to remove the generosity to keep the maximum scalability. Today we found that if $K = 1$, the analysis can be done in time ($<$ 50 ms) for a system having less than 700 components, which covers largely the real industrial scenarios.

That is also why we have developed another method[25] (each component is an HMM, compared with this method where each observed variable is an HMM). The benchmark shows that the component-based method is less accurate (4–11% decrease), but can be computed very fast, so is more scalable for industrial usage.

A question may be raised: why no parallel computation? If we introduce parallel computation, will it solve the performance problem that we can re-enable the generosity of the approach? Effectively, each HMM can be computed independently once their parameter has been defined. But this can only be done on a PC. Because finally we need to embed this online analysis function into an embedded partition, and current standard prevents the parallel computation inside the same partition (even between partitions it is not recommended). Therefore we have to ignore this option.

6.5. Discussion: modeling method of multiple HMMs

In our work, the Viterbi algorithm from Section 5.7 is applied for each observation variable independently for the related HMM and then these results are combined using voting. It seems that a single HMM where all observations are combined would be better as this would allow us to use the combined observation to detect the right faulty component and not only trust some majority of observations. It is very possible that a majority of the observed variables are not reflecting a particular fault and, therefore, it is impossible to conclude where the failure source is on their own. We will discuss this problem in the following two parts.

First, why multiple HMMs instead of one HMM? This concern is fully understandable and in fact it was indeed our modeling choice in our earlier experiments. The problem is related to HMM's characteristics: the smaller the difference between the values of different matrix elements is, the worse its prediction would be (entropy is smaller). It is more likely to happen if observation number increases (e.g. if we take 0.1 as granularity, for a HMM having 1 inner state, if it contains 2 observation states, it only has 1/11 chance ([0.5, 0.5] among the following 11 possible emission matrices [0, 1], [0.1, 0.9], …, [0.9, 0.1], [1, 0])) that these observations cannot distinguish from each other from the view point of information. But if it contains 3 observation states, it can have a chance of 18/66 ([0.1,0.1,0.8], [0.3,0.3,0.4], etc.)

So, when combining all observations into a single HMM, the whole HMM's parameter (particularly for emission matrix, because initial and transition matrices are the same for all variables) is ''flattened''. The difference between elements of emission matrix is decreased. This leads to a non-accurate fault isolation result (around 70% according to our tests). So the choice of multiple HMMs has improved this single HMM modeling.

Second, will a partial vote mislead the result? The answer is first a yes. Let us consider a extreme case: there is only one observation variable available (so only one HMM will do the vote), and the faulty component will not influence this variable at all (i.e. not direct/indirect dataflow towards this variable), the conclusion given by this HMM will make no sense. The final answer is that we can avoid this. All variables are not necessarily to be controlled, but at least we need to guarantee that if we take a backward dependency analysis, all components must be covered at least once by the controlled variable set.

7. Conclusion

This article presents an HMM-based approach to online diagnose accidental faults for real-time embedded systems. By introducing reasonable and appropriate abstraction of complex systems, HMM is used to describe the healthy or faulty states of a system's hardware components. The observation sequences are derived from the test results with respect to the functional constraints defined in system specifications. They are parametrized to statistically simulate the real system's behavior. As it is not easy to obtain rich accidental fault data from a real system, the Baum–Welch algorithm cannot be employed here to train the parameters in HMMs. The parameters in the initial state distribution and the state transition matrix are computed using the failure rates of hardware components. The parameters in the emission matrix are estimated using the failure propagation algorithm. The estimation method is based on the principles of FTA and the maximum entropy in Bayesian probability theory. A fault propagation distribution is thus computed, whose parameters are adapted using the backward algorithm and observations. The parameterized HMMs are then used to online diagnose accidental faults using a vote algorithm integrated with a low-pass filter. We have designed a specific test bed to analyze the measures including the sensitivity, specificity, precision, accuracy and F1-score by generating a large amount of test cases. The test results show that the proposed approach is robust, efficient and accurate.

Funding

This work is supported by the Japan NII Scholarship and the French ministries of Industry and Research and the Midi-Pyrénées regional authorities through the ITEA2 OPEES project.

References

1. Avizienis A, Laprie JC, Randell B, et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Depend Secure Comput* 2004; 1(1): 11–33.
2. Avizienis A, Laprie JC, Randell B, et al. *Fundamental concepts of dependability*. Technical report series, University of Newcastle upon Tyne, Computing Science, 2001.
3. Madan BB, Goševa-Popstojanova K, Vaidyanathan K, et al. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Perform Eval* 2004; 56(1): 167–186.
4. Gertler J. *Fault detection and diagnosis in engineering systems,*1st edn. Boca Raton, FL: CRC Press, 1998.
5. Venkatasubramanian V, Rengaswamy R, Yin K, et al. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Comput Chem Eng* 2003; 27(3): 293–311.
6. Isermann R and Balle P. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Eng Practice* 1997; 5(5): 709–719.
7. Isermann R. Model-based fault-detection and diagnosis—status and applications. *Ann Rev Control* 2005; 29(1): 71–85.
8. Nakajima S, Furukawa S and Ueda Y. Co-analysis of SYSML and Simulink models for cyber-physical systems design. In *2nd workshop on cyber-physical systems, networks, and applications*, pp. 473–478.
9. Lee EA. Cyber physical systems: Design challenges. In *11th IEEE international symposium on object oriented real-time distributed computing (ISORC)*. IEEE, pp. 363–369.
10. Rabiner L and Juang BH. An introduction to hidden Markov models. *IEEE ASSP Magazine* 1986; 3(1): 4–16.
11. Rabiner L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE* 1989; 77(2): 257–286.
12. Elliott RJ, Aggoun L and Moore JB. *Hidden Markov Models: Estimation and Control (Applications of Mathematics)*. New York: Springer, 1995.
13. Baum LE. An equality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities* 1972; 3: 1–8.
14. Viterbi AJ. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans Informat Theory* 1967; 13(2): 260–269.
15. Clarke EM and Zuliani P. Statistical model checking for cyber-physical systems. In Bultan T and Hsiung PA (eds), *Automated Technology for Verification and Analysis*. New York: Springer, 2011, pp. 1–12.
16. Boutros T and Liang M. Detection and diagnosis of bearing and cutting tool faults using hidden markov models. *Mech Syst Signal Process* 2011; 25(6): 2102–2124.
17. Tobon-Mejia DA, Medjaher K, Zerhouni N, et al. A data-driven failure prognostics method based on mixture of gaussians hidden Markov models. *IEEE Trans Reliab* 2012; 61(2): 491–503.
18. Smyth P. Hidden Markov models for fault detection in dynamic systems. *Pattern Recognition* 1994; 27(1): 149–164.
19. Ying J, Kirubarajan T, Pattipati KR, et al. A hidden Markov model-based algorithm for fault diagnosis with partial and imperfect tests. *IEEE Trans Syst Man Cybernet C* 2000; 30(4): 463–473.
20. Hertz J, Krogh A and Palmer R. *Introduction to the theory of neural computation* (*Advanced book program*, vol. 1). Reading, MA: Addison-Wesley, 1991.
21. Silverman BW. *Density estimation for statistics and data analysis* (*Chapman & Hall/CRC Monographs on Statistics & Applied Probability*, vol. 26). Boca Raton, FL: CRC Press, 1986.

22. Rosenblatt M. A central limit theorem and a strong mixing condition. *Proc Natl Acad Sci USA* 1956; 42(1): 43.
23. Ge N, Nakajima S and Pantel M. Efficient online analysis of accidental fault localization for dynamic systems using hidden Markov model. In *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium*. Society for Computer Simulation International, p. 16.
24. Fielding AH and Bell JF. A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environ Conserv* 1997; 24(01): 38–49.
25. Ge N, Nakajima S and Pantel M. Hidden Markov model based automated fault localization for integration testing. In *4th IEEE international conference on software engineering and service science (ICSESS)*. IEEE, pp. 184–187.

Author biographies

Ning Ge received her PhD in 2014 from the Université de Toulouse, France, working at IRIT (Institut de Recherche en Informatique de Toulouse). She is currently a research engineer at IRT Saint-Exupéry, France.

Shin Nakajima is a full professor at National Institute of Informatics, Tokyo, Japan. He is also an adjunct professor at SOKENDAI and a visiting professor at Tokyo Institute of Technology, Tokyo, Japan.

Marc Pantel is an associate professor at the Université de Toulouse in France, at the IRIT (Institut de Recherche en Informatique de Toulouse) in the ACADIE team (Assistance to the Certification of Critical Embedded or Distributed Applications). He got his PhD in Computer Science from Université de Toulouse in 1994 and 2 engineering degrees in computer science 1990 and electrical engineering 1989. He conducts research in the integration of formal methods and model-based system engineering for safety critical systems in relation with industrial partners from the aerospace domain.