Fault diagnosis based on identified discrete-event models

Marcos Moreira, Jean-Jacques Lesage

Fault diagnosis based on identified discrete-event models

Marcos V. Moreira[a], Jean-Jacques Lesage[b]

[a] *Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica, 21949-900, Rio de Janeiro, R.J., Brazil (e-mail: moreira.mv@poli.ufrj.br)*

[b] *LURPA, ENS Paris-Saclay, Univ. Paris-Sud, Université Paris-Saclay, 94235 Cachan, France (e-mail: jean-jacques.lesage@ens-paris-saclay.fr)*

Abstract

Fault diagnosis of Discrete-Event Systems consists of detecting and isolating the occurrence of faults within a bounded number of event occurrences. Recently, a new model for discrete-event system identification with the aim of fault detection, called Deterministic Automaton with Outputs and Conditional Transitions (DAOCT), has been proposed in the literature. The model is computed from observed fault-free paths, and represents the fault-free system behavior. In order to obtain compact models, loops are introduced in the model, which implies that sequences that are not observed can be generated leading to an exceeding language. This exceeding language is associated with possible non-detectable faults, and must be reduced in order to use the model for fault detection. After detecting the fault occurrence, its isolation is carried out by analyzing residuals. In this paper, we present a fault diagnosis scheme based on the DAOCT model. We show that the proposed fault diagnosis scheme is more efficient than other approaches proposed in the literature, in the sense that the exceeding language can be drastically reduced, reducing the number of non-detectable fault occurrences, and, in some cases, reducing also the delay for fault diagnosis. A practical example, consisting of a plant simulated by using a 3D simulation software controlled by a Programmable Logic Controller, is used to illustrate the results of the paper.

Keywords: Fault diagnosis, System identification, Discrete-event systems, Finite automata, Black-box identification.

1. Introduction

The problem of fault diagnosis, *i.e.*, the detection and isolation of faults, has received considerable attention from the scientific community over the last years. In [27], a discrete-event approach for fault diagnosis is introduced, and since then, several works have been proposed for fault detection and isolation, and also for the verification of diagnosability of the system, *i.e.*, the capability of identifying the occurrence of a fault event within a bounded number of event occurrences [11, 22, 20, 8, 9, 7, 6, 5, 28]. In all these works, it is assumed that the complete system behavior is known, *i.e.*, the system behavior before and after the occurrence of fault events.

Although methods for fault diagnosis based on the complete system behavior can be successfully applied to small systems, they are difficult to be implemented on large and complex systems for the following reasons: *(i)* in general, large automated systems are composed of several components, whose models and interactions between these models, are difficult or even impossible to be obtained; *(ii)* the modeling process requires engineers that know the complete plant behavior, and are also familiar with discrete-event modeling techniques; *(iii)* the post-fault behavior of the system is difficult to be predicted due its size and complexity; and *(iv)* only predefined faults can be detected by the diagnoser computed considering the complete behavior of the system.

Since expert building of behavioral models is error-prone and highly time consuming, an alternative way is to obtain a model by identification. Analogously to continuous systems identification techniques [1, 17], identification methods for DES aims at yielding a mathematical model which closely approximates the actual system behavior, from data observing during the system functioning. In the case of DES, the data observed during the system functioning are sequences of binary events, and identified models are abstract machines, like Petri nets or finite automata. Several works in the literature propose identification methods based on automata or Petri nets for different purposes [18, 4, 13, 12, 14, 2, 15]. The majority of these works address the problem of identifying Petri net models that are not suitable for fault diagnosis. A method for the identification of a Petri net model suitable for fault diagnosis is proposed in [3]. In [3], the faulty behavior of the system is identified based on the observation of the events generated by the system, and it is assumed that the fault-free model is known. Thus, the method proposed in [3] does not address the problem of obtaining large and complex fault-free models of DES.

Fault diagnosis techniques based on an identified fault-free model of the system have been proposed in [16], [25], [24], and [26]. In these works, the two main ideas are: *(i)*

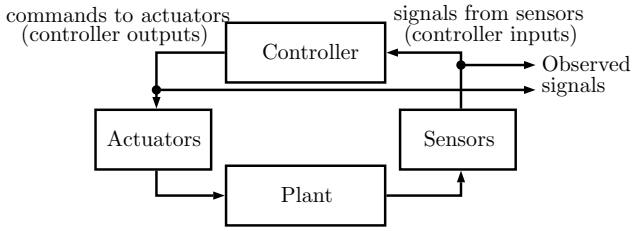

Figure 1: Closed-loop discrete-event system

to automate the process of obtaining the fault-free model of the system by using identification; and *(ii)* when a fault has been detected through a discrepancy between the system behavior and the model, to use a technique based on residuals for fault isolation.

In [16], a monolithic model for fault detection, that is capable of representing the behavior of a closed-loop system, is proposed. This model is non-deterministic with state outputs, and has been called Non-Deterministic Autonomous Automaton with Output (NDAAO). The NDAAO is obtained from observed sequences of binary signals exchanged between the plant and the controller (sensor signals emitted by the plant and actuator commands generated by the controller), as shown in Figure 1. In [16], it is shown that the identified NDAAO generates all observed sequences of signals used in the identification process. Furthermore, a trade-off between size and accuracy of the identified model can be found thanks to an adequate adjustment of the parametric algorithm used for identification. Indeed, for reducing the size of the model, equivalent states are merged, what introduces loops in the NDAAO, generating sequences that have not been observed. This exceeding language can increase the number of non-detectable faults of the system, and may prevent the fault detection scheme to be implemented. In order to deal with this trade-off, in [16], a free parameter k is used to compute the NDAAO, and it is shown that the NDAAO is $k+1$-complete in the sense of [19], *i.e.*, a sequence of signals of length smaller than or equal to $k+1$ belongs to the identified NDAAO if, and only if, it is observed in the system.

In [25] and [26] the fault detection strategy proposed in [16] is extended to systems with a high degree of concurrency. As in [16], the NDAAO is used, and the same trade-off between model size and accuracy is observed in these works. Moreover, a fault isolation strategy is proposed based on the computation of residuals that are associated with the observed changes of signals that are not expected by the model, and the changes that are expected but are not observed in the system. From the residuals, possible justifications for the fault occurrence are obtained, reducing the effort of maintenance of the faulty equipments.

Recently, in [21], a new model for discrete-event system identification, that is more efficient for fault detection than the model presented in [16, 25, 26], called Deterministic Automaton with Outputs and Conditional Transitions (DAOCT), has been proposed. The exceeding language generated by the DAOCT is reduced in comparison with the exceeding language generated by the NDAAO, due to a path estimation function that is added to the model. In particular, if the identified DAOCT does not have cyclic paths, then there is no exceeding language. As in [16], in [21] it is assumed that the binary input and output signals of the controller are measured, generating the observed fault-free paths of the system. Using this information, the DAOCT is computed. The DAOCT also satisfies the property of $k+1$-completeness, if sequences of observed signals are considered, or, equivalently, k-completeness if sequences of events are considered.

In this paper, we propose a fault diagnosis scheme based on the DAOCT model presented in [21]. We show that the proposed scheme is more efficient than the methods presented in [16] and [24], for the monolithic case, in the sense that the exceeding language of the DAOCT model obtained by using our fault detection strategy can be greatly reduced in comparison with the exceeding language obtained by using the NDAAO model. This leads to a reduction in the number of non-detectable fault occurrences, improving the efficiency of the fault detection method. Since the fault detection strategy proposed in this work uses more information about the fault-free behavior of the system than the method proposed in [16] and [24], then the delay for fault detection can also be reduced. In addition, due to the use of some specific information about the observed paths in the fault detection scheme, the residuals can be refined, indicating more precisely the possible justifications to isolate the fault. A practical example, consisting of a plant simulated using a 3D simulation software and controlled by a Programmable Logic Controller, is used to illustrate the results of the paper.

This paper is organized as follows. In Section 1.1, we present the contributions of the paper with respect to preliminary results obtained in other works. In Section 2, we present some preliminary concepts and the basic ideas of fault diagnosis based on the fault-free behavior of the system. In Section 3, we formulate the problem of system identification with the aim of fault detection, and in Section 4, we introduce the DAOCT model for system identification. In Section 5 we present the fault detection scheme, and in Section 6, we present the fault isolation method. In Section 7, we present a practical example to illustrate the results of the paper. Finally, in Section 8, the conclusions are drawn.

1.1. Preliminary results

In [21], the DAOCT model is introduced for fault detection, and the language generated by the DAOCT is defined based on the feasible events of the states of the model and on a path estimation function that estimates the observed paths that may have been executed after the transposition of each transition of the model. Then, in [21], it is shown that the exceeding language generated by

the DAOCT model can be drastically reduced in comparison with the exceeding language of the NDAAO model, which shows that the DAOCT model is more suitable for fault detection than the NDAAO model. However, since the DAOCT model performs a path estimation after the observation of each event executed by the system, it must be reinitialized all time when it reaches its initial state. The reinitializability of the model is essential for the use of the DAOCT model for fault detection, and this problem has not been addressed in [21]. In this paper, we define the reinitializability property of the model, and we present conditions to verify if the identified model is reinitializable.

A fault detection algorithm using the DAOCT model is also not presented in [21]. In this paper, we present the fault detection algorithm, and it is shown that two new conditions for fault detection can be introduced, reducing even more the exceeding language of the DAOCT model presented in [21]. This fact shows that the use of the DAOCT model with the fault detection algorithm proposed in this paper can be more efficient for fault detection than it was supposed in [21].

The new fault detection scheme proposed in this paper can also lead to a more accurate fault isolation. In order to show this fact, we use the fault isolation scheme proposed in [24] with our method for fault detection, and we show that the fault isolation has been improved using the DAOCT model with the new fault detection algorithm. It is important to remark that fault isolation is not addressed in [21].

Finally, differently from [21], we present a practical example simulated in a virtual plant simulation software to illustrate the fault diagnosis method proposed in the paper.

2. Preliminaries

2.1. Notation and Definitions

Let $G = (X, \Sigma, f, x_0, X_m)$ denote a deterministic automaton [10], where X is the set of states, Σ is the finite set of events, $f : X \times \Sigma^\star \to X$ is the transition function, where $\Sigma^\star$ is the Kleene-closure of Σ, x_0 is the initial state of the system, and X_m is the set of marked states.

The language generated by G is defined as $L(G) = \{s \in \Sigma^\star : f(x_0, s)!\}$, where ! denotes *is defined*. The prefix-closure of a language L is defined as $\overline{L} = \{s \in \Sigma^\star : (\exists t \in \Sigma^\star)[st \in L]\}$. Note that the language generated by G is prefix-closed by definition.

The function of feasible events $\Gamma : X \to 2^\Sigma$, is defined as $\Gamma(x) = \{\sigma \in \Sigma : f(x, \sigma)!\}$.

The set of all subsequences of a sequence $s \in \Sigma^\star$ is defined as $Sub(s) = \{w \in \Sigma^\star : (\exists t, w, v \in \Sigma^\star)(s = twv)\}$.

A path p of an automaton G is a sequence of states and events that can be executed by the system, *i.e.*, a path $p = (x_1, \sigma_1, x_2, \sigma_2, \ldots, \sigma_{l-1}, x_l)$ is feasible in G if, and only if, $x_i \in X$, for $i = 1, 2, \ldots, l$, $\sigma_i \in \Sigma$, for $i = 1, 2, \ldots, l-1$, and $f(x_i, \sigma_i) = x_{i+1}$, $i = 1, \ldots, l-1$. The length of a path is defined as the number of vertices in the path, and is denoted here as $\|p\|$. Thus, $\|p\| = l$. A path is said to be cyclic if $x_l = x_1$.

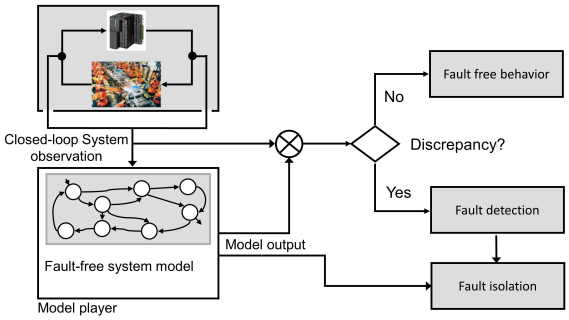

Figure 2: Model-based diagnosis based on a fault-free model.

Let P be a set of paths, and define function $\psi : P \to \Sigma^\star$, that extracts from a path $p \in P$, the sequence of events associated with p. Thus, if $p = (x_1, \sigma_1, x_2, \sigma_2, \ldots, \sigma_{l-1}, x_l)$, then $\psi(p) = \sigma_1\sigma_2 \ldots \sigma_{l-1}$.

The length of a sequence of events $s \in \Sigma^\star$ is denoted as $|s|$.

The set of non-negative integers is denoted by $\mathbb{N}$, and the set formed only with 0 and 1 is denoted by $\mathbb{N}_1 = \{0, 1\}$.

The difference between two sets A and B is denoted by $A \setminus B$.

2.2. Fault diagnosis based on the fault-free behavior of the system

In order to deal with the problem of fault diagnosis of large automated systems, whose complete behavior can be very difficult or even impossible to be obtained, mainly the post-fault behavior, some works in the literature propose the identification of the fault-free behavior of the system. The fault diagnosis system compares the sequences of events or input/output (I/O) vectors formed of the signals of sensors and actuators, and declares the occurrence of a fault when there is a discrepancy between the observed behavior and the predicted behavior described by the identified model, as shown in Figure 2. After the fault has been detected, a comparison between the observed signals and the expected signals according to the model is carried out, generating residuals, that are used to isolate the fault.

In this paper, we propose a fault diagnosis scheme based on the identified fault-free behavior of the system. In addition, differently from the traditional approaches that provide necessary and sufficient conditions for the diagnosability of fault events, in this paper we address the problem of diagnosing fault occurrences, since the occurrence of a fault in a given system state can be detectable, while the

occurrence of the same fault event in a different state can be non-detectable. Indeed, we cannot address the diagnosability verification problem since we do not identify or model the post-fault behavior of the system, and we also do not specify which fault must be detected. The main advantage of this approach is that not only predefined faults can be detected, but any fault that leads the system to execute a sequence of events that has not been observed in the fault-free behavior.

In the fault diagnosis strategy proposed in this paper, fault isolation is performed after the fault has been detected, by analyzing the history of sequences of events executed by the system and the I/O vector of sensors and actuators.

3. Discrete-event system identification with the aim of fault detection

Let us consider the closed-loop system depicted in Figure 1, and assume that the controller has m_i binary input signals, i_h, for $h = 1, \ldots, m_i$, and m_o binary output signals, o_h, for $h = 1, \ldots, m_o$. Let vector

$$u(t_1) = \left[\begin{array}{cccccc} i_1(t_1) & \ldots & i_{m_i}(t_1) & o_1(t_1) & \ldots & o_{m_o}(t_1) \end{array} \right]^T,$$

denote the observation of the controller signals at time instant t_1. Thus, vector $u(t_1)$ represents the I/O vector of the system at a given time instant t_1. As the system evolves, the I/O vector of the system may change due to changes in sensor readings or actuator commands. Let us consider that there is a change in at least one of the variables of u. Then, at the time instant immediately after this change, t_2, a new vector $u(t_2)$ is observed. Since, in this paper, we consider only untimed system models, we may define the instantaneous changes in the values of the controller signals as the system events, σ, and represent the I/O vector of the system $u(t_j)$, by u_j. Thus, the transition from one vector of controller signals u_1 to another vector u_2, is represented by the transition (u_1, σ, u_2). If a sequence of l vectors of controller signals, and the corresponding changes in these signals, is observed, we have an observed path of the system $p = (u_1, \sigma_1, u_2, \sigma_2, \ldots, \sigma_{l-1}, u_l)$.

The goal of system identification is to find a model that is capable of describing the observed behavior of the system. Let us consider that the observed paths of the system are denoted as $p_i = (u_{i,1}, \sigma_{i,1}, u_{i,2}, \sigma_{i,2}, \ldots, \sigma_{i,l_i-1}, u_{i,l_i})$, for $i = 1, \ldots, r$, where r is the number of observed paths, and l_i is the number of vertices of each path p_i. Let us also assume that all paths start at the same vertex, *i.e.*, all I/O vectors $u_{i,1}$, for $i = 1, \ldots, r$, are equal, and that some paths can be cyclic, *i.e.*, $u_{i,l_i} = u_{i,1}$ for $i \in \{1, \ldots, r\}$, representing that the system is cyclic. Thus, associated with each path p_i there is a sequence of events $s_i = \psi(p_i) = \sigma_{i,1}\sigma_{i,2}\ldots\sigma_{i,l_i-1}$, where $\psi : P \to \Sigma^\star$ with $P = \{p_1, \ldots, p_r\}$, and a sequence of output vectors $\omega_i = u_{i,1}u_{i,2}\ldots u_{i,l_i}$. The following assumption is considered in the paper.

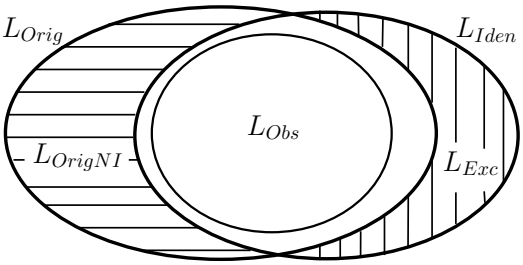

Figure 3: Relation between the languages L_{Orig}, L_{Iden}, L_{Exc}, and L_{OrigNI}.

A1. None of the paths p_i has an associated sequence of events $s_i = \psi(p_i)$ that is a prefix of the sequence of events of another path p_j, $s_j = \psi(p_j)$, where $i \neq j$, and that all cyclic paths can occur repeatedly in the system, and in any order.

The following definition of the language observed by the system can be stated:

$$L_{Obs} := \bigcup_{i=1}^{r} \overline{\{s_i\}}. \tag{1}$$

Since the objective of system identification is to find a model that simulates the observed behavior described by L_{Obs}, then the language generated by the identified model, L_{Iden}, must satisfy $L_{Obs} \subseteq L_{Iden}$. This relation between L_{Obs} and L_{Iden} is depicted in the diagram of Figure 3.

In a finite time, only part of the sequences of events that the system can generate can be observed, which means that $L_{Obs} \subset L_{Orig}$, where L_{Orig} denotes the never known language generated by the system. The relation between the observed language and the original language generated by the system is also described in the diagram of Figure 3.

As it can be seen in Figure 3, two other languages can be defined: (*i*) $L_{Exc} = L_{Iden} \setminus L_{Orig}$; and (*ii*) $L_{OrigNI} = L_{Orig} \setminus L_{Iden}$. L_{Exc} represents the sequences of events that can be generated by the identified automaton but do not belong to the original system behavior. Since the fault detection strategy is based on the observation of events and comparison with the sequences generated by the model, if a sequence of events that is not in the original fault-free system is observed and is in the language of the identified model, then the fault occurrence is not detected. Thus, L_{Exc} are formed of faulty sequences that cannot be detected by the fault detection system. On the other hand, L_{OrigNI} is associated with the sequences that are in the original fault-free system, but are not identified because the paths associated with these sequences have not been observed. The sequences of events of L_{OrigNI} are associated with false alarms generated by the fault detection system. Clearly, both languages must be reduced in order to obtain an efficient fault detection scheme.

In [16], it is shown that if a sufficiently large number of controller vectors are observed, then there exists a number $n_0 \in \mathbb{N}$ such that the difference $L_{Orig}^{\leq n_0} \setminus L_{Obs}^{\leq n_0} \approx \emptyset$, where $L_{Orig}^{\leq n_0}$ and $L_{Obs}^{\leq n_0}$ denote the sets formed of all sequences of events of length smaller than or equal to n_0 of L_{Orig} and L_{Obs}, respectively. Thus, since $L_{Obs} \subseteq L_{Iden}$, the subset of L_{OrigNI} formed of all sequences of events of length smaller than or equal to n_0, $L_{OrigNI}^{\leq n_0}$, is also approximately the empty set. Let us assume that $L_{OrigNI}^{\leq n_0} = \emptyset$, which happens if all paths of length smaller than or equal to $n_0 + 1$, or equivalently all sequences of length smaller than or equal to n_0, have been observed (see Figure 3). Then, all sequences of events of length smaller than or equal to n_0 that does not belong to the identified model are faulty sequences, and the fault detection system will not raise false alarms. This assumption is formalized as follows.

A2. All paths of length $n_0 + 1$ of the original system are observed, and, consequently, $L_{OrigNI}^{\leq n_0} = \emptyset$.

The goal of Assumption **A2** is to reduce the number of false alarms raised by the fault detection system. It is important to remark that this assumption is not restrictive for the fault detection method proposed in the paper, *i.e.*, even if **A2** is not true, the fault detection algorithm can still be used.

In [21], the definition of k-completeness based on sequences of events is presented. In order to present this definition, let us first define the set of all observed paths $P := \{p_i : i \in R\}$, where $R = \{1, 2, \ldots, r\}$, and the language formed of all observed subsequences of events of length n, as follows:

$$L_{S,Obs}^n := \{s \in \Sigma^\star : (|s| = n)[\exists i \in R, s \in Sub(\psi(p_i))]\},$$

where $\psi : P \to \Sigma^\star$.

Definition 1. *A model is said to be k-complete if for all $n \leq k$, $L_{S,Obs}^n = L_{S,Iden}^n$, where $L_{S,Iden}^n$ is the set formed of all subsequences of events of the identified model of length n.* □

In the next section, the model proposed in [21] for the identification of DES with the aim of fault diagnosis is presented.

4. Deterministic Automaton with Outputs and Conditional Transitions

In [21], a modified automaton model that is suitable for fault diagnosis is proposed. The modified automaton is deterministic, with a state output function, and the transitions must satisfy a condition to be transposed associated with the observed paths used to construct the model. This automaton is called Deterministic Automaton with Outputs and Conditional Transitions (DAOCT), and is formally defined as follows.

Definition 2. *A Deterministic Automaton with Outputs and Conditional Transitions (DAOCT) is the eight-tuple:*

$$DAOCT = (X, \Sigma, \Omega, f, \lambda, R, \theta, x_0),$$

where X is the set of states, Σ is the set of events, $\Omega \subset \mathbb{N}_1^{m_i+m_o}$ is the set of I/O vectors, $f : X \times \Sigma^\star \to X$ is the deterministic transition function, $\lambda : X \to \Omega$, is the state output function, $R = \{1, 2, \ldots, r\}$ is the set of path indices, $\theta : X \times \Sigma \to 2^R$ is the path estimation function, and x_0 is the initial state. □

The sets of events and I/O vectors associated with each observed path p_i, $i = 1, \ldots, r$, are denoted in this paper, respectively, as Σ_i and Ω_i. Thus, the set of events and the set of I/O vectors of the identified model are, respectively, $\Sigma = \cup_{i=1}^r \Sigma_i$ and $\Omega = \cup_{i=1}^r \Omega_i$.

The DAOCT is obtained from the observed paths p_i, $i = 1, \ldots, r$, and, as in [16], a free parameter k is used to construct the identified model. In order to do so, it is first computed modified paths p_i^k from paths p_i such that the vertices of p_i^k are sequences of I/O vectors of length at most equal to k as follows:

$$p_i^k = (y_{i,1}, \sigma_{i,1}, y_{i,2}, \sigma_{i,2}, \ldots, \sigma_{i,l_{i-1}}, y_{i,l_i}), \tag{2}$$

where

$$y_{i,j} = \begin{cases} (u_{i,j-k+1}, \ldots, u_{i,j}), & \text{if } k \leq j \leq l_i \\ (u_{i,1}, \ldots, u_{i,j}), & \text{if } j < k \end{cases}. \tag{3}$$

Note that the sequence of events of p_i^k is equal to the sequence of events of path p_i. Thus, the unique difference between p_i and p_i^k is that each vertex of p_i^k is now associated with a sequence of vectors instead of a single I/O vector. According to the algorithm for the construction of the identified model presented in [21], each state $x \in X$ of the DAOCT is associated with a different vertex of the paths p_i^k, such that $|X| = \sum_{i=1}^r l_i$. The labeling function $\tilde{\lambda} : X \to \Omega^k$, where Ω^k is formed of all sequences of symbols of Ω of length smaller than or equal to k, is used in [21] to associate to each state $x \in X$, a vertex of one of the paths p_i^k. Then, the output $\lambda(x)$ is defined for each state $x \in X$ as the last I/O vector of $\tilde{\lambda}(x)$.

Each transition $x' = f(x, \sigma)$ of automaton DAOCT has a corresponding set $\theta(x, \sigma)$ of indices that is associated with the paths p_i that contain transition (x, σ, x'). Function θ is used in the DAOCT evolution rule to provide a path estimator, such that if the paths associated with a transition are not coherent with the paths of the observed sequence of events, then the transition is not enabled. This fact is clearly presented in the definition of the language generated by the DAOCT. In order to present the language generated by the DAOCT, it is first necessary to extend the domain of function θ to consider the execution of sequences of events, obtaining the extended path estimation function $\theta_s : X \times \Sigma^\star \to 2^R$. θ_s can be

defined recursively as:

$$\theta_s(x,\epsilon) = R,$$
$$\theta_s(x,s\sigma) = \begin{cases} \theta_s(x,s) \cap \theta(x',\sigma), & \text{where } x' = f(x,s), \\ & \text{if } f(x,s\sigma)! \\ \text{undefined, otherwise.} \end{cases} \tag{4}$$

The language generated by the DAOCT is given by

$$L(\text{DAOCT}) := \{s \in \Sigma^\star : f(x_0,s)! \wedge \theta_s(x_0,s) \neq \emptyset\}. \tag{5}$$

Note that a sequence of events $s \in \Sigma^\star$ is only feasible in the DAOCT, if $f(x_0,s)$ is defined, and there is at least one path in the path estimate after the occurrence of s, represented by condition $\theta_s(x_0,s) \neq \emptyset$.

Example 1. *Let us consider a system with three binary controller signals, and let us consider the observation of three paths p_i, $i = 1,\ldots,3$, given as:*

$$p_1 = \left(\begin{bmatrix}1\\0\\0\end{bmatrix}, a, \begin{bmatrix}1\\1\\0\end{bmatrix}, b, \begin{bmatrix}0\\1\\1\end{bmatrix}, c, \begin{bmatrix}0\\0\\0\end{bmatrix}, d, \begin{bmatrix}0\\0\\1\end{bmatrix}, e, \begin{bmatrix}1\\0\\0\end{bmatrix} \right),$$

$$p_2 = \left(\begin{bmatrix}1\\0\\0\end{bmatrix}, g, \begin{bmatrix}0\\0\\0\end{bmatrix}, h, \begin{bmatrix}1\\1\\0\end{bmatrix}, b, \begin{bmatrix}0\\1\\1\end{bmatrix}, c, \begin{bmatrix}0\\0\\0\end{bmatrix}, i, \begin{bmatrix}1\\0\\0\end{bmatrix}, j, \begin{bmatrix}0\\1\\1\end{bmatrix}, l, \begin{bmatrix}1\\0\\0\end{bmatrix} \right),$$

$$p_3 = \left(\begin{bmatrix}1\\0\\0\end{bmatrix}, g, \begin{bmatrix}0\\0\\0\end{bmatrix}, h, \begin{bmatrix}1\\1\\0\end{bmatrix}, b, \begin{bmatrix}0\\1\\1\end{bmatrix}, i, \begin{bmatrix}1\\1\\1\end{bmatrix}, m, \begin{bmatrix}0\\0\\0\end{bmatrix}, d, \begin{bmatrix}0\\0\\1\end{bmatrix}, n, \begin{bmatrix}1\\1\\0\end{bmatrix} \right),$$

where each event is associated with the rising or the falling edge of the controller signals. For instance, a, denotes the rising edge of the second controller signal, and b, denotes the simultaneous falling edge of the first controller signal and the rising edge of the third controller signal.

Let us now compute identified models obtained according to the method presented in [21]. In Figures 4 and 5, we present the identified models, $DAOCT_1$ and $DAOCT_2$, obtained by choosing $k = 1$ and $k = 2$, respectively. Note that each transition is labeled with an event from Σ, and a set associated with the paths p_i where each transition is defined, i.e., each transition (x,σ,x'), where $x' = f(x,\sigma)$, of the identified model is labeled with σ and $\theta(x,\sigma)$. In addition, notice that $DAOCT_2$ is acyclic.

In order to illustrate the reduction in the exceeding language by using the path estimation function θ_s, let us consider that sequence $s = abi$ has been observed. Note that $s \notin L_{Obs}$. Let us also consider that Assumption **A2** *is valid for $n_0 = 3$, i.e., $L_{Orig}^{\leq 3} = L_{Obs}^{\leq 3}$. Then, if $s \in L(DAOCT)$, s belongs to the exceeding language L_{Exc}. Note that transition function $f(x_0,s)$ is defined in both models $DAOCT_1$ and $DAOCT_2$ of Figures 4 and 5, respectively, which implies that it belongs to both identified models without considering the path estimation function. However, since $\theta_s(x_0,s) = \emptyset$ for both models $DAOCT_1$ and $DAOCT_2$, then $s \notin L(DAOCT_1)$ and $s \notin L(DAOCT_2)$.* □

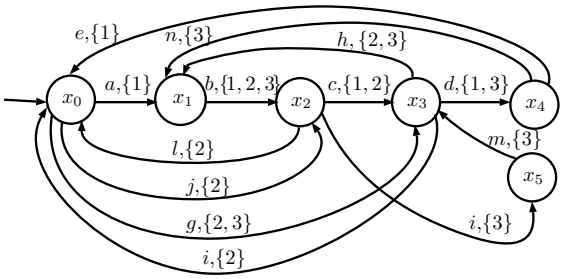

Figure 4: DAOCT_1 computed in Example 1 considering $k = 1$.

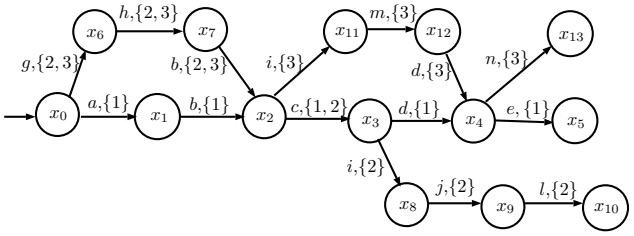

Figure 5: DAOCT_2 computed in Example 1 considering $k = 2$.

In [21] it is shown that the DAOCT model simulates the observed language of the system L_{Obs}, as stated in the following theorem.

Theorem 1. $L_{Obs} \subseteq L(DAOCT)$.

Proof. See [21]. □

In order to show that the identified DAOCT model is k-complete, let us first define the language formed of all subsequences of events of length n generated by the DAOCT as follows:

$$L_S^n(\text{DAOCT}) := \{s \in \Sigma^\star : (|s| = n)[\exists x_i \in X, f(x_i,s)! \wedge \theta_s(x_i,s) \neq \emptyset]\}.$$

Then, the following result can be stated [21].

Theorem 2. *For a given value of k, the identified DAOCT is k-complete, i.e., $L_S^n(DAOCT) = L_{S,Obs}^n$, for all $n \leq k$.*

Proof. See [21]. □

Theorems 1 and 2 show that the DAOCT model is suitable for fault detection, since it simulates the observed language, and any subsequence of length k belongs to the DAOCT model if, and only if, it has been observed, *i.e.*, the approximation of the observed language given by the identified model can be made more accurate increasing the value of k. However, the increase in the value of k, leads to models with a higher number of states. Since a path estimation function is used in the DAOCT model, in [21] it is shown that it is possible to obtain more compact systems using the DAOCT model than using the NDAAO model proposed in [16], with less exceeding language. Thus, the enhanced model proposed in [21] reduces the number of

non-detectable faults in comparison with the NDAAO. In the next theorem we show that if the DAOCT does not have cyclic paths, then $L_{Exc} = \emptyset$.

Theorem 3. *If the identified DAOCT does not have cyclic paths for a given value of k, then $L_{Exc} = \emptyset$.*

Proof. See [21]. □

Let us introduce the language generated by the DAOCT formed of all traces of length smaller than or equal to a given value n as follows:

$$L^{\leq n}(\text{DAOCT}) := \left(\bigcup_{i=0}^{n} L_S^i(\text{DAOCT}) \right) \cap L(\text{DAOCT}).$$

According to Theorem 3, if k is chosen such that the DAOCT does not have cyclic paths, then, $L_{Exc} = L(\text{DAOCT}) \setminus L_{Orig} = \emptyset$, and there is no non-detectable faults. In addition, if Assumption **A1** also holds, the observed language $L_{Obs}^{\leq n_0}$ is equal to the original system language $L_{Orig}^{\leq n_0}$, and there is no false alarms for all observed traces of length smaller than or equal to n_0. Thus, under both assumptions, $L^{\leq n_0}(\text{DAOCT}) = L_{Orig}^{\leq n_0}$.

In the next section we introduce the fault detection method based on the DAOCT model, and we show that the exceeding language associated with the identified model can be reduced by considering two other conditions associated with the counting of event observations.

5. Fault detection scheme

In this section, we show how to use the identified DAOCT model for fault detection. As shown in Theorem 1, the DAOCT simulates language L_{Obs} formed of all sequences of events observed in the identification procedure. Thus, as long as the events of a sequence $s = \psi(p_i)$, where p_i is an observed path of the system, are executed by the system, the fault detector observes the events, and plays the model following the behavior of s. After sequence $s = \psi(p_i)$ has been observed, the model is reinitialized and a new sequence can be played by the fault detector. Let us call the sequence of events that is played by the fault detector without reinitializing the model as a model run. Thus, the fault detector must evaluate if the current model run corresponds to one of the sequences in L_{Obs}. If the system generates an event that is not expected in the model, the fault is detected.

We show in the sequel that it is possible to reduce the exceeding language associated with the identified model by using additional information provided by the fault-free paths used in the identification process.

Example 2. *Let us consider the identified model $DAOCT_1$, depicted in Figure 4, computed from the fault-free paths p_1, p_2 and p_3, presented in Example 1, considering $k = 1$. We show in the sequel two cases in which it is possible to identify sequences in language $L(DAOCT_1)$ that do belong to the observed language L_{Obs}:*

1) Note that each path p_1, p_2, and p_3 of Example 1 can be distinguished from the other paths after a bounded number of event observations. Let n_i denote the minimum number of event observations such that path p_i can be distinguished from all other paths p_j, $i \neq j$, and $j = 1, 2, 3$. In this example, we have that $n_1 = 1$, $n_2 = 4$, and $n_3 = 4$. Let us consider now that sequence $s = gi$ has been observed. Note that $f(x_0, s)$ is defined and that $\theta_s(x_0, s) = \{2\}$. Thus, according to Equation (5), $s \in L(DAOCT_1)$. However, since the minimum number of event observations to uniquely identify path p_2 is equal to $n_2 = 4$, and only two events have been observed, then we are certain that path p_2 is not being executed by the closed-loop system. Thus, we are certain that $s \notin L_{Obs}$, and a fault occurrence can be detected by using this information.

2) Note that each path p_i must reach its last vertex u_{i,l_i} after the observation of $l_i - 1$ events. Let us consider path p_2. In this case, we have that $u_{2,l_2} = \lambda(x_0)$, and $l_2 = 8$, which means that state x_0 must be reached after 7 event observations if path p_2 is observed. Suppose now that sequence $s = ghbchbc$ has been observed. Note that $f(x_0, s)$ is defined and $\theta_s(x_0, s) = \{2\}$. Thus, $s \in L(DAOCT_1)$. However, from the transition diagram of Figure 4, it can be seen that state x_3 is reached after the occurrence of s, and not state x_0 as expected. Thus, we are certain that path p_2 has not been executed by the system, and a fault occurrence can be detected by using this information. □

Example 2 shows that it is possible to derive two conditions associated with the observed fault-free paths used in the identification process, that can be easily checked by counting the number of event occurrences. In the first case, the minimum number n_i of event observations to distinguish the observed path p_i from the other paths p_j, $j \in R$ and $i \neq j$, is used. It is important to remark that, since it is assumed that each trace $s_i = \psi(p_i)$ cannot be a prefix of another trace $s_j = \psi(p_j)$, where $i \neq j$ and $i, j \in R$, then there always exists a number $0 < n_i < l_i$ associated with each path p_i. Thus, if path p_i is wrongly estimated as the unique possible path before n_i event occurrences, then the fault is detected. In the second case, if the final vertex y_{i,l_i} of the estimated path p_i is not reached after $l_i - 1$ event occurrences, then the path estimate is wrong and the fault has occurred. Based on these observations, we may state the following definition.

Definition 3. *Let $s \in \Sigma^\star$ be a model run such that $x = f(x_0, s)$. Then, an event $\sigma \in \Sigma$ is said to be viable in state $x \in X$ of the DAOCT model, if it satisfies the following four conditions:*

C1. $\sigma \in \Gamma(x)$;

C2. $\theta_s(x_0, s\sigma) \neq \emptyset$;

C3. *If $|\theta_s(x_0, s)| > 1$ and $\theta_s(x_0, s\sigma) = \{i\}$, then $|s\sigma| \geq n_i$;*

C4. *If $|s\sigma| = l_i - 1$, for $i \in \theta_s(x_0, s\sigma)$, then $\tilde{\lambda}(x') = y_{i,l_i}$, where $x' = f(x, \sigma)$, or there exists $j \in \theta_s(x_0, s\sigma)$ such that $|s\sigma| < l_j - 1$.* □

Conditions **C1** and **C2** guarantee that $s\sigma \in L(\text{DAOCT})$. If Condition **C3** is not true, then path p_i is identified before the minimum number n_i of events that must be observed in order to estimate it. Thus, a fault has occurred. Finally, if Condition **C4** is not true, then the length of the observed trace $s\sigma$ is equal to the maximum length among all sequences of the estimated paths in $\theta_s(x_0, s\sigma)$, without reaching the final vertex of any of these paths, which implies that a fault has occurred.

The basic idea of the fault detection scheme is to compare the viable events of the identified fault-free model with the observed events. If the observed event does not satisfy conditions **C1**-**C4** to be viable, then the fault is detected. In the sequel, we present the fault detection algorithm. In order to do so, it is necessary to define sets $V = \{(i, y_{i,l_i}) : i \in R\}$, and $N = \{(i, n_i) : i \in R\}$.

Algorithm 1. *Fault detection algorithm*

Input: *Identified DAOCT model, $\tilde{\lambda}$, V, N.*
Output: *Fault detection.*

1: *Define the current state of the DAOCT model $x_c = x_0$*
2: *Define the current path estimate $\theta_{s,c} = \{1, 2, \ldots, r\}$*
3: *Define the counter of event observations $c = 0$*
4: *Wait for the next event observation $\sigma \in \Sigma$*
5: $c \leftarrow c + 1$.
6: **if** $\sigma \notin \Gamma(x_c)$ **then**
7: *Fault detected*
8: *Stop the algorithm*
9: **end if**
10: *Define $\theta_{s,n} = \theta_{s,c} \cap \theta(x_c, \sigma)$*
11: **if** $\theta_{s,n} = \emptyset$ **then**
12: *Fault detected*
13: *Stop the algorithm*
14: **end if**
15: **if** $|\theta_{s,n}| = 1$ *and* $|\theta_{s,c}| > 1$ **then**
16: *Find the pair $(i, n_i) \in N$ such that $\theta_{s,n} = \{i\}$*
17: **if** $n_i > c$ **then**
18: *Fault detected*
19: *Stop the algorithm*
20: **end if**
21: **end if**
22: *Define state $x_n = f(x_c, \sigma)$*
23: *Define set $\Lambda = \{l_i : i \in \theta_{s,n}\}$*
24: **if** *there exists $l_i \in \Lambda$ such that $c = l_i - 1$, $\tilde{\lambda}(x_n) = y_{i,l_i}$, and $u_{i,l_i} = u_{i,1}$* **then**
25: $\theta_{s,c} \leftarrow \{1, 2, \ldots, r\}$
26: $x_c \leftarrow x_0$
27: *Go to line 3*
28: **end if**
29: **if** *there exists $l_i \in \Lambda$ such that $c = l_i - 1$, $\tilde{\lambda}(x_n) = y_{i,l_i}$, and $u_{i,l_i} \neq u_{i,1}$* **then**
30: $x_c \leftarrow x_n$
31: $\theta_{s,c} \leftarrow \emptyset$
32: *Go to line 4*
33: **end if**
34: **if** $\max_{l_j \in \Lambda} l_j = c + 1$ **then**
35: *Fault detected*
36: *Stop the algorithm*
37: **end if**
38: *Define $\theta'_{s,n} = \{i \in \theta_{s,n} : l_i = c + 1\}$*
39: $\theta_{s,n} \leftarrow \theta_{s,n} \setminus \theta'_{s,n}$
40: $x_c \leftarrow x_n$
41: $\theta_{s,c} \leftarrow \theta_{s,n}$
42: *Go to line 4*

The fault detection procedure works as follows. Lines 1 to 3 initializes the current state x_c of the DAOCT model to x_0, the current path estimate $\theta_{s,c}$ to R, and an integer variable c, that counts the number of event observations, to 0. In line 4 we wait for the next event observation $\sigma \in \Sigma$, and when it occurs, in line 5 we update the value of counter c. In lines 6 to 9, Condition **C1** of Definition 3 is verified, and if σ is not in the feasible event set $\Gamma(x_c)$ of the DAOCT, the fault is detected and the procedure stopped. In lines 10 to 14, Condition **C2** is verified, and the fault is detected if the observed sequence of events does not belong to any path p_i used in the construction of the DAOCT model, *i.e.*, $\theta_{s,n} = \emptyset$. In lines 15 to 21,

Condition **C3** of Definition 3 is checked. Since each path p_i, for $i = 1, \ldots, r$, can be distinguished after the occurrence of n_i events from all other paths p_j, $j \neq i$, then if p_i is the unique path in the path estimation function θ_s before n_i event observations, then it corresponds to an non-viable sequence of observed events, and the fault is detected. In lines 22 to 28, the model is reinitialized if one of the estimated paths p_i^k, associated with a cyclic path p_i, where $i \in \theta_s(x_c, \sigma)$, has reached its final vertex y_{i,l_i}, after the observation of $l_i - 1$ events. In order to do so, it is verified if for counter $c = l_i - 1$, the sequence of output vectors associated to x_n, $\tilde{\lambda}(x_n)$, is equal to y_{i,l_i}, and if $u_{i,l_i} = u_{i,1}$. The information of the sequences y_{i,l_i}, for $i = 1, \ldots, r$, is obtained from set V. In lines 29 to 33, it is verified if a path p_i^k associated with a non-cyclic path p_i has reached its final state, *i.e.*, the system has reached a predicted deadlock. If it is true, then the current state x_c is updated to x_n, and the path estimate $\theta_{s,c}$ is defined as the empty set. If any event is observed after that, then the fault occurrence is detected by Condition **C1** of lines 6 to 9, or by Condition **C2** of lines 10 to 14. In lines 34 to 37, the test of Condition **C4** is carried out, and the fault is detected if the number of event observations is equal to the number of observations of the estimated path with the longest associated trace. Since, in lines 24 to 28, the reinitialization of the DAOCT model is performed returning to line 3, and in lines 29 to 33, the model reaches the final state of a non-cyclic path returning to line 4, if line 34 is reached this means that the model has not been reinitialized or reached a deadlock, which implies that the final state associated with the longest length path has not been reached. Finally, in lines 38 to 42, $\theta_{s,c}$ and x_c are updated and the procedure returns to line 4, waiting for the next event observation.

It is important to remark that it is assumed here that if $s = \psi(p_i^k)$ is observed, then path p_i^k must be uniquely determined by following the method proposed in Algorithm 1. This assumption is necessary for the correct reinitialization of the model in lines 22 to 28, and is formally defined as follows.

Definition 4. *Let $s = \psi(p_i^k)$, for $i \in \{1, 2, \ldots . r\}$. Then, the DAOCT model is said to be reinitializable if there does not exist $s' \in \overline{\{s\}}$ of length $|s'| = l_j - 1$, where $j \in \theta_s(x_0, s')$ and $l_j < l_i$, such that $x' = f(x_0, s')$, and $\tilde{\lambda}(x') = y_{j,l_j}$.* □

Let us consider that path p_i is executed by the system, and let $s = \psi(p_i)$. In this case, if the condition of Definition 4 is not satisfied, then there exists a trace $s' \in \overline{\{s\}}$, such that a state $x' = f(x_0, s')$ of the DAOCT, whose associated sequence of I/O vectors is $\tilde{\lambda}(x') = y_{j,l_j}$, is reached. Since $|s'| = l_j - 1$, then according to lines 24 and 29 of Algorithm 1, the model is reinitialized or stopped after the observation of s', and path p_i is not necessarily played in the model.

In the sequel, we present sufficient conditions that guarantee the reinitializability of the identified DAOCT model.

Theorem 4. *If the DAOCT is acyclic, then it is reinitializable.*

Proof. If the DAOCT is acyclic, then, according to the algorithm for its construction presented in [21], the intersection between the path estimates of two transitions leaving the same state of the DAOCT is empty. Thus, two paths $i, j \in \theta_s(x_0, s)$ if, and only if, s is a prefix of both traces $s_i = \psi(p_i^k)$ and $s_j = \psi(p_j^k)$. Since, by assumption, s_j cannot be a prefix of s_i, for any $i, j \in \{1, \ldots, r\}$, then if the last vector of I/O signals y_{j,l_j} of path p_j^k is equal to vertex y_{i,l_j} of path p_i^k, the path estimate after executing the prefix of s_i, s_i', that reaches state x such that $\tilde{\lambda}(x) = y_{i,l_j}$ does not contain j, *i.e.*, $j \notin \theta_s(x_0, s_i')$. Thus, according to Definition 4, the DAOCT is reinitializable. □

Theorem 5. *If the final vertex of path p_i^k, y_{i,l_i}, is not equal to any vertex y_{j,l_i}, for all $i, j \in \{1, \ldots, r\}$, $i \neq j$, then the DAOCT is reinitializable.*

Proof. The proof is straightforward from Definition 4. □

It is important to remark that even if Theorems 4 and 5 are not satisfied, then the DAOCT model can be reinitializable, as it is shown in the following example.

Example 3. *Let us consider the same paths p_i, $i = 1, 2, 3$, presented in Example 1. If we consider $k = 1$, then the identified DAOCT is cyclic, and therefore does not satisfy the condition of Theorem 4. In addition, the last vector $y_{1,6}$ of path p_1^1 is equal to vector $y_{2,6}$ of path p_2^1. Thus, the condition of Theorem 5 is also not satisfied. However, since the path estimate after the observation of the first event cannot contain both paths 1 and 2, then the DAOCT is reinitializable.* □

If the conditions of Theorems 4 and 5 are not true, then the reinitializability of the identified DAOCT model can be easily checked by playing in the model the paths p_i, for $i = 1, \ldots, r$, following the steps of Algorithm 1, and verifying if it is capable of reinitializing correctly. If the DAOCT model is reinitializable, then it can be used for fault detection.

It is important to remark that since, by assumption, the trace associated with an observed path cannot be a prefix of the trace associated with another observed path, then there always exists a value of k such that the condition of Theorem 5 is satisfied. Consequently, if the DAOCT is not reinitializable, then one can always choose a greater value of k and obtain a reinitializable DAOCT model.

According to Algorithm 1, the fault detection scheme is based on the playing of the DAOCT model, following the observation of the events generated by the system. Thus, we can define a fault detection function $FD : \Sigma^\star \to \mathbb{N}_1$ as follows:

$$FD(s) = \begin{cases} 1, \text{ if a fault is detected using Algorithm 1} \\ \quad \text{after the observation of } s, \\ 0, \text{ otherwise.} \end{cases} \tag{6}$$

When a state associated with the end of a cyclic path p_i^k is reached after the observation of a trace $s \in \Sigma^\star$, and $FD(s) = 0$, then the model is reinitialized, *i.e.*, the current state is set to x_0 and the current path estimator is set to R, and a new run of the DAOCT model is carried out according to the observed events. Thus, a fault cannot be detected only if the event observations after the fault occurrence are viable in the model, and the system can be reinitialized or reaches a deadlock before the fault is detected. Consequently, only observed output vectors of the fault-free behavior can be reached according to the fault detector scheme without detecting the fault. Thus, I/O vectors associated with dangerous configurations for the system and its operators can be detected as soon as they are observed, and actions can be executed to avoid damages to the system.

In the following example, we show the reduction in the exceeding language considering the viable sequences of events of the DAOCT model in comparison with the exceeding language generated by the NDAAO model proposed in [16]. In order to do so, let us first define the language formed of all viable sequences of length smaller than or equal to n generated by the DAOCT:

$$L^{\leq n}_{\widetilde{ND},DAOCT} = \{s \in \Sigma^\star : (|s| \leq n)[FD(s) = 0]\}. \quad (7)$$

Thus, we can define the exceeding language generated by the model with respect to Algorithm 1 as $L^{\leq n}_{\widetilde{Exc},DAOCT} = L^{\leq n}_{\widetilde{ND},DAOCT} \setminus L^{\leq n}_{\widetilde{Orig}}$. Let also $L^{\leq n}_{\widetilde{Exc},NDAAO}$ denote the set formed of all sequences of length smaller than or equal to n in the exceeding language generated by using the method proposed in [16]. It is important to remark that we assume here that $n \leq n_0$. Then, according to Assumption **A1**, $L^{\leq n}_{\widetilde{Obs}} = L^{\leq n}_{\widetilde{Orig}}$, for $n \leq n_0$.

Example 4. *Let us consider the DAOCT model depicted in Figure 4, obtained for $k = 1$ from the paths presented in Example 1. In Figure 6, we compare the cardinality of the exceeding language $L^{\leq n}_{\widetilde{Exc},DAOCT}$, for $n = 1, \ldots, 7$, of the DAOCT model considering conditions* **C1**-**C4** *for the viability of the observed traces ($\diamond$), with the cardinality of the exceeding language $L^{\leq n}_{\widetilde{Exc},NDAAO}$, obtained by using the method proposed in [16] ($*$). As it can be seen from Figure 6, there is a huge reduction in the exceeding language by using the four conditions* **C1**-**C4**, *with only 18 traces in $L^{\leq n}_{\widetilde{Exc},DAOCT}$ for $n = 7$. This leads to a reduction of the non-detectable fault occurrences by using the method proposed in Algorithm 1 in comparison with the method proposed in [16].* □

It is important to remark that, since in the fault detection strategy proposed in this paper we use more information than the method proposed in [16], then the delay for fault detection, defined as the number of event observations between the occurrence of the fault event and its detection, can also be reduced. This improves the quality of the fault detection scheme in the sense that a fast detection can avoid damages caused by the fault due to an incorrect system behavior.

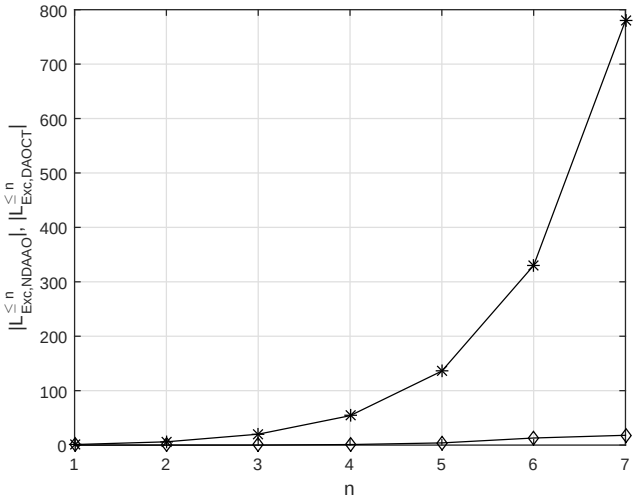

Figure 6: Comparison between the cardinality of the exceeding language $L^{\leq n}_{\widetilde{Exc},NDAAO}$, generated by the NDAAO model (*), and the cardinality of the exceeding language $L^{\leq n}_{\widetilde{Exc},DAOCT}$, generated by the DAOCT model using the fault detection scheme presented in Algorithm 1 ($\diamond$), for different values of n.

Remark 1. *Since the fault detection is based on playing the DAOCT model after the observation of events executed by the system, then the computational complexity of Algorithm 1 is linear in the size of the DAOCT for each model run.* □

6. Fault isolation scheme

In [26], a method for fault isolation based on residuals is proposed. The main idea is to identify which signal changes have occurred after the fault detection, and to compare these changes with the possible signal changes predicted by the model. Thus, the residuals are used to capture fault symptoms that help in determining which sensor, or actuator, or part of the plant is possibly affected. In this paper, the same reasoning is deployed to isolate the faults. The difference between the residuals proposed in this paper and the one presented in [26], is that, since we use more conditions to detect a fault occurrence than the method proposed in [16], then the residuals proposed here will always lead to a smaller set of possible faults in the system. Thus, we can isolate more precisely the fault using the DAOCT model than using the NDAAO model proposed in [16].

In order to introduce the four residuals proposed in this work, let us first denote by $u_k(i)$ the *i-th* signal of the vector of I/O signals u_k. Then the rising edge (resp. falling edge) of the *i-th* signal is detected when $u_k(i) = 0$ (resp. $u_k(i) = 1$), and $u_j(i) = 1$ (resp. $u_j(i) = 0$), for the sequence of observed vectors $u_k u_j$. Let $u^f(i)$, $u^r(i)$, and $u^n(i)$ denote, respectively, the falling edge, the rising edge, and the no change in value of the *i-th* signal of the

sequence of I/O vectors $u_k u_j$. Then, we can define the edge function as follows [26]:

$$Edge(u_k(i)u_j(i)) = \begin{cases} u^f(i), \text{ if } u_k(i) = 1 \wedge u_j(i) = 0, \\ u^r(i), \text{ if } u_k(i) = 0 \wedge u_j(i) = 1, \\ u^n(i), \text{ if } u_k(i) = u_j(i). \end{cases} \tag{8}$$

We can also define the set formed of all signal changes observed in the sequence of I/O vectors $u_k u_j$ as follows [26]:

Definition 5. *(Evolution set)*

$$ES(u_k u_j) = \bigcup_{i=1}^{m} \{Edge(u_k(i)u_j(i)) : Edge(u_k(i)u_j(i)) \neq u^n(i)\}$$

where m is the number of I/O signals of vectors u_k and u_j.

Note that after the occurrence of a trace $s \in \Sigma^\star$ in the DAOCT model such that $FD(s) = 0$, a unique state $\tilde{x} \in X$ is reached. From state $\tilde{x}$, we can analyze the next states of the model in order to find those that satisfy the four conditions **C1**-**C4** of Definition 3, *i.e.*, we can search for a viable event $\sigma \in \Sigma$. This procedure leads to a set of states that can be reached from $\tilde{x}$ that do not lead to the fault detection. In the sequel, we present the reachable states function $RS : \Sigma^\star \to 2^X$ that provides all states of the DAOCT model that can be reached from a state $\tilde{x}$ that is reached after the observation of a trace $s \in \Sigma^\star$.

Definition 6. *(Reachable states)*

$$RS(s) = \{x \in X : (\exists \sigma \in \Sigma)[FD(s\sigma) = 0, x = f(x_0, s\sigma)]\}.$$

Using Definitions 5 and 6, we can now define the four residuals for fault isolation. In all residuals, $\tilde{x} = f(x_0, s)$ denotes the last state of the model such that the fault has not been detected, *i.e.*, $FD(s) = 0$, and u denotes the I/O observed vector that led to the fault detection.

Definition 7. *(Residuals for fault isolation)*

$$Res_1(\tilde{x}, u, s) = ES(\lambda(\tilde{x})u) \setminus \bigcup_{\forall x' \in RS(s)} ES(\lambda(\tilde{x})\lambda(x')),$$

$$Res_2(\tilde{x}, u, s) = ES(\lambda(\tilde{x})u) \setminus \bigcap_{\forall x' \in RS(s)} ES(\lambda(\tilde{x})\lambda(x')),$$

$$Res_3(\tilde{x}, u, s) = \bigcap_{\forall x' \in RS(s)} ES(\lambda(\tilde{x})\lambda(x')) \setminus ES(\lambda(\tilde{x})u),$$

$$Res_4(\tilde{x}, u, s) = \bigcup_{\forall x' \in RS(s)} ES(\lambda(\tilde{x})\lambda(x')) \setminus ES(\lambda(\tilde{x})u).$$

The interpretation of the residuals is equal to the one presented in [26]. The unique difference is that, in this work, we are searching for the states reachable from $\tilde{x}$ in the DAOCT model, taking into account Conditions **C1**-**C4** for fault detection, implemented in Algorithm 1. This naturally leads to a more precise fault isolation.

In the first residual, Res_1, the observed signal changes that are not predicted by any of the possible next states of the model are computed. In Res_2, the signal changes that are observed, and are not one of the unavoidable signal changes that should be observed according to the model are computed. Note that, by definition, $Res_1(\tilde{x}, u, s) \subseteq Res_2(\tilde{x}, u, s)$, for any $\tilde{x} \in X, u \in \mathbb{N}_1^m, s \in \Sigma^\star$. Residual Res_3 computes all signal changes that should be observed according to the model, and have not been observed. And finally, Res_4 computes all signal changes that are possible in any of the predicted next states of the model, and have not been observed. By definition, $Res_3(\tilde{x}, u, s) \subseteq Res_4(\tilde{x}, u, s)$, for any $\tilde{x} \in X, u \in \mathbb{N}_1^m, s \in \Sigma^\star$.

The residuals are used in the next section to isolate the faults simulated in a sorting unit system. In order to do so, we use the same strategy presented in [26], *i.e.*, we first analyze the changes in vector u associated with residuals Res_1 and Res_3, and if the fault cannot be isolated, we analyze the other signal changes presented in Res_2 and Res_4. It is important to remark that, in some cases, the fault symptom can be an unexpected change in a sensor signal, but the fault occurred in an actuator that is directly related to that sensor. Thus, in order to correctly isolate faults, it is necessary to group actuators with the sensors that are directly influenced by them. This case is analyzed in the following section.

7. Practical Example

A sorting unit system is depicted in Figure 7. The objective of this system is to sort parcels, that are randomly delivered to the feeder conveyor (FC), according to their size. Two sensors, located at the end of the feeder conveyor, k_1 and k_2, inform the presence of a parcel and its corresponding size. If the parcel is a small one then the falling edge of sensor k_1 is observed without observing $k_2 = 1$, and if the parcel is a big one, we observe $k_1 = 1$ and $k_2 = 1$. The first pusher in the distribution conveyor (DC), Pusher 1 (P_1), send small parcels to the first slide, and big parcels are sent to the second slide by Pusher 2 (P_2). When the distribution conveyor has a parcel, and another parcel arrives at the end of the feeder conveyor ($k_1 = 1$), the feeder conveyor is stopped and is turned on again only when the parcel of the distribution conveyor is sorted. When a small (resp. big) parcel is in front of Pusher 1 (resp. Pusher 2), detected by the falling edge of the signal of the sensor located at the side of Pusher 1 (resp. Pusher 2), sensor k_3 (resp. k_4), the distribution conveyor is turned off, and is turned on again only after the end of the movement of Pusher 1 (resp. Pusher 2). Each pusher has two sensors to indicate if it is completely

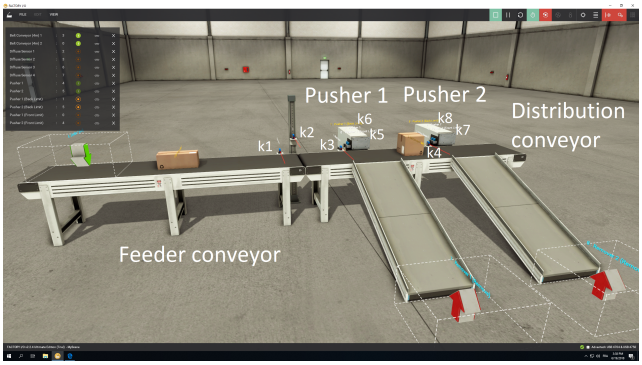

Figure 7: Sorting unit system simulated using software FACTORY I/O

retracted or extended, sensors k_5 and k_6 for Pusher 1, and sensors k_7 and k_8 for Pusher 2. Thus, the controller of this system has 8 inputs (corresponding to the 8 sensors) and 4 outputs (corresponding to the 4 actuators), which implies that each I/O vector u_j has 12 binary entries, defined as follows:

$$u_j = [\ k_7\ \ k_8\ \ k_5\ \ k_6\ \ k_2\ \ k_1\ \ k_4\ \ k_3\ \ P_2\ \ P_1\ \ FC\ \ DC\ ]^T.$$

In order to obtain the fault-free behavior of the system depicted in Figure 7, simulations have been carried out using the software FACTORY I/O [23], guaranteeing that the system is completely free of faults. Based on the simulation of the fault-free behavior of the plant, thirteen cyclic fault-free paths p_i, $i = 1, \ldots, 13$, have been identified, and we have computed the DAOCT and the NDAAO models for $k = 1$ and $k = 2$.

The number of states of the DAOCT is 34 and 51 for $k = 1$ and $k = 2$, respectively, while the number of states of the NDAAO is 34 and 40 for $k = 1$ and $k = 2$, respectively. Although the NDAAO leads to more compact models than the DAOCT for the same value of k, the exceeding language is larger using the NDAAO model, which implies that the DAOCT model is more suitable for fault detection than the NDAAO model.

In order to analyze the efficiency of Algorithm 1 for fault detection, 42 permanent and intermittent fault occurrences in sensors and actuators have been simulated. In all 42 scenarios, we simulated the fault of only one sensor or actuator at the same time, and, in some cases, the same fault had been simulated more than once, generating a different scenario to evaluate if it can be detected depending on when the fault occurs. In Table 1, we show the number of detected and undetected fault occurrences by using the NDAAO model for $k = 1$, and the DAOCT model for $k = 1$ and $k = 2$. Note that the DAOCT model can detect 35 fault occurrences, against 29 faults detected using the NDAAO model. Among the six fault occurrences detected by the method presented in Algorithm 1, and not detected by using the method of [16], two have been detected by the condition associated with the non-viability of the estimated paths (Condition **C2**, lines 10 to 14 of Algorithm 1), three by the condition associated with the identification of the path before the correct number of event observations (Condition **C3**, lines 15 to 21 of Algorithm 1), and one associated with a non-expected state reached after the observation of the maximum number of event observations among all estimated paths (Condition **C4**, lines 34 to 37 of Algorithm 1). The other 29 fault occurrences have been identified by the occurrence of an unfeasible event of the DAOCT model (Condition **C1**, lines 6 to 9 of Algorithm 1). It is also important to remark that two fault occurrences have been detected by using the method proposed in this paper with a smaller delay than by using the method presented in [16]. This shows that the fault detection strategy proposed in this paper can detect the fault occurrence faster than the method proposed in [16].

From Table 1, it can be seen that the efficiency of the method proposed in this paper is 83.3%, against 69% of the method proposed in [16]. In addition, from Table 1, it can be seen that the detection method using the DAOCT model with $k = 2$ has the same efficiency than the model obtained using $k = 1$, *i.e.*, the same seven fault occurrences remains non-detectable increasing the value of k. Since the DAOCT model is acyclic for $k = 2$, its exceeding language, $L^{\leq n}_{Exc,DAOCT}$, is empty for all values of n. Thus, we can conclude that the behavior of the system after the occurrence of the undetected faults has observation equal to the observation of fault-free paths or leads the model to a deadlock, and, consequently, are non-detectable for any value of k. Thus, although the exceeding language of the DAOCT model obtained for $k = 1$ is not empty, it has been capable of detecting the fault occurrences for all simulated cases with the same efficiency than the DAOCT model obtained for $k = 2$.

In order to isolate the faults, the four residuals have been computed for all 35 faults detected by the DAOCT model considering $k = 1$. We have also grouped the actuators with the sensors that are directly affected by these actuators as follows: (i) Feeder Conveyor with sensors k_1 and k_2; (ii) Distribution Conveyor with sensors k_3 and k_4; (iii) Pusher 1 with sensors k_5 and k_6; and (iv) Pusher 2 with sensors k_7 and k_8.

As an example of the use of the residuals for fault isolation, let us consider the simulation of a fault in Pusher 1. We have considered that, for some reason, after the third event observation, Pusher 1 is stuck always extended. The observed sequence of I/O vectors, starting at the initial

state of the system, is given by:

$$\begin{bmatrix} 0\\1\\0\\1\\0\\1\\0\\0\\0\\0\\1\\1 \end{bmatrix} \underset{\rightarrow}{u^r(5)} \begin{bmatrix} 0\\1\\0\\1\\1\\1\\0\\0\\0\\0\\1\\1 \end{bmatrix} \underset{\rightarrow}{u^f(6)} \begin{bmatrix} 0\\1\\0\\1\\1\\1\\0\\0\\0\\0\\1\\1 \end{bmatrix} \underset{\rightarrow}{u^f(5)} \begin{bmatrix} 0\\1\\0\\1\\0\\0\\0\\0\\0\\0\\1\\1 \end{bmatrix} \underset{\rightarrow}{u^f(4)} \begin{bmatrix} 0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\1\\1 \end{bmatrix}$$

where the first four I/O vectors correspond to the output λ of states 1, 2, 3 and 4, respectively, of the DAOCT model depicted in Figure 8. The last I/O vector does not correspond to any possible next state of the model and violates Condition **C1**, leading to the fault detection according to Algorithm 1. In Figure 8 we have omitted the events, and we have presented only the paths associated with each transition. Symbol $i : j$, where $i, j \in R$, in the path estimate set of each transition denotes the set formed of all paths from i to j. Note that the last state $\tilde{x} = 4$ of the DAOCT model that does not lead to fault detection has output vector equal to:

$$\lambda(\tilde{x}) = [\ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ ]^T,$$

and the observed vector u, that is not predicted by the model, is given by:

$$u = [\ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ ]^T.$$

The predicted outputs of the states that follow state $\tilde{x}$ in the DAOCT model are:

$$\lambda(5) = [\ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ ]^T,$$

$$\lambda(21) = [\ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ ]^T,$$

$$\lambda(22) = [\ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ ]^T,$$

$$\lambda(28) = [\ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ ]^T.$$

In this case, the residuals are given by:

$$Res_1(\tilde{x}, u, s) = Res_2(\tilde{x}, u, s) = \{u^f(4)\}, Res_3(\tilde{x}, u, s) = \emptyset,$$

$$Res_4(\tilde{x}, u, s) = \{u^r(7), u^r(8), u^r(9), u^f(12)\}.$$

From $Res_1(\tilde{x}, u, s)$, we obtain the observed signal changes that were not expected in any of the next states of the DAOCT model. Since $u^f(4)$ is associated with the falling edge of sensor k_6, then we must check if sensor k_6 has a malfunctioning, or Pusher 1, associated with k_6, is faulty. Since, in this case, Pusher 1 is stuck extended, we were able to correctly isolate the fault.

The fault has been isolated by considering the actuators and their associated sensors in 31 of the 35 cases for which the fault could be detected using Algorithm 1. In the four cases where the residuals were not sufficient to isolate the fault, several transitions had occurred after the fault occurrence, which makes fault isolation more difficult. In these cases, more information from the system is needed to correctly isolate the fault. Thus, by using the method proposed in this paper, we were able to detect 35 fault occurrences, and make the correct diagnosis of 31 of the 42 fault occurrences simulated in the system.

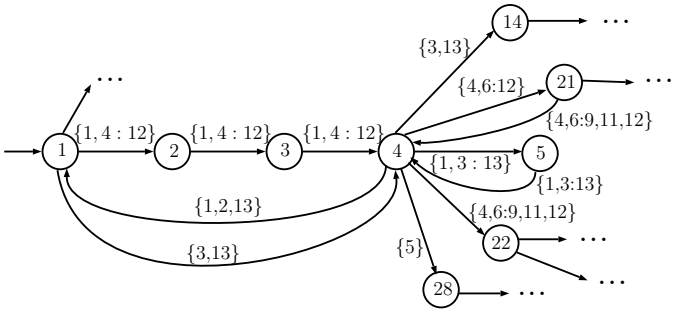

Figure 8: Part of the DAOCT model obtained for $k = 1$ of the sorting unit system.

Table 1: Efficiency of the fault detection scheme

	NDAAO ($k = 1$)	DAOCT ($k = 1$)	DAOCT ($k = 2$)
Detected faults	29	35	35
Non-detected faults	13	7	7
Efficiency (%)	69%	83.3%	83.3%

8. Conclusions

We present in this paper a fault diagnosis method based on an identified DAOCT model. Since the exceeding language of the DAOCT is reduced using the fault detection scheme, then it is more suitable for fault detection than other methods proposed in the literature. A fault isolation method, based on residuals computed using the fault detection scheme is also presented, and a practical example is used to illustrate the efficiency of the proposed method. We are currently investigating the use of the DAOCT model for distributed identification of DES.

Acknowledgments

The work of M. V. Moreira was partially supported by the Brazilian Research Council (CNPq) under grants 305267/2018-3 and 431307/2018-0, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) Finance Code 001.

References

[1] Astrom, K. J., Eykhoff, P., 1971. System identification: a survey. Automatica 7, 123–162.

[2] Cabasino, M. P., Darondeau, P., Fanti, M. P., Seatzu, C., 2013. Model identification and synthesis of discrete-event systems, in Contemporary Issues in System Science and Engineering. Wiley.

[3] Cabasino, M. P., Giua, A., Hadjicostis, C. N., Seatzu, C., 2014. Fault model identification and synthesis in petri nets. Discrete Event Dynamic Syst. 24 (3), 275–307.

[4] Cabasino, M. P., Giua, A., Seatzu, C., 2007. Identification of petri nets from knowledge of their language. Discrete Event Dynamic Syst. 17 (4), 447–474.

[5] Cabral, F. G., Moreira, M. V., 2017. Synchronous codiagnosability of modular discrete-event systems. In: IFAC World Congress. IFAC, Toulouse, France, pp. 7025–7030.

[6] Cabral, F. G., Moreira, M. V., Diene, O., 2015. Online fault diagnosis of modular discrete-event systems. In: Decision and Control (CDC), 2015 IEEE 54th Annual Conference on. IEEE, pp. 4450–4455.

[7] Cabral, F. G., Moreira, M. V., Diene, O., Basilio, J. C., 2015. A Petri net diagnoser for discrete event systems modeled by finite state automata. IEEE Transactions on Automatic Control, 59–71.

[8] Carvalho, L. K., Basilio, J. C., Moreira, M. V., 2012. Robust diagnosis of discrete-event systems against intermittent loss of observations. Automatica 48 (9), 2068–2078.

[9] Carvalho, L. K., Moreira, M. V., Basilio, J. C., Lafortune, S., 2013. Robust diagnosis of discrete-event systems against permanent loss of observations. Automatica 49 (1), 223–231.

[10] Cassandras, C., Lafortune, S., 2008. Introduction to Discrete Event System. Springer-Verlag New York, Inc., Secaucus, NJ.

[11] Debouk, R., Lafortune, S., Teneketzis, D., 2000. Coordinated decentralized protocols for failure diagnosis of discrete event systems. Discrete Event Dynamic Systems: Theory and Applications 10 (1), 33–86.

[12] Estrada-Vargas, A. P., Lesage, J.-J., López-Mellado, E., 2014. A stepwise method for identification of controlled discrete manufacturing systems. Int. J. Comput. Integr. Manuf. 28 (2), 187–199.

[13] Estrada-Vargas, A. P., López-Mellado, E., Lesage, J.-J., 2010. A comparative analysis of recent identification approaches for discrete-event systems. Math. Probl. Eng. 2010, 1–21.

[14] Estrada-Vargas, A. P., López-Mellado, E., Lesage, J.-J., 2014. Input-output identification of controlled discrete manufacturing systems. Int. J. Syst. Sci. 45 (3), 456–471.

[15] Estrada-Vargas, A. P., López-Mellado, E., Lesage, J.-J., 2015. A black-box identification method for automated discrete event systems. IEEE Transactions on Automation Science and Engineering 14 (3), 1321–1336.

[16] Klein, S., Litz, L., Lesage, J.-J., 2005. Fault detection of discrete event systems using an identification approach. In: 16th IFAC World Congress. Prague, Czech Republic, pp. 92–97.

[17] Ljung, L., 1999. System Identification: Theory for the User, 2nd Edition. Prentice Hall.

[18] Medhi, S. O. E., Leclercq, E., Lefebvre, D., 2006. Petri nets design and identification for the diagnosis of discrete event systems. In: IAR Annu. Meeting. IEEE.

[19] Moor, T., Raisch, J., Young, S., 1998. Supervisory control of hybrid systems via l-complete approximations. In: Proceedings of the IEEE Workshop on Discrete-Event Systems. Cagliari, Italy, pp. 426–431.

[20] Moreira, M. V., Jesus, T. C., Basilio, J. C., 2011. Polynomial time verification of decentralized diagnosability of discrete event systems. IEEE Transactions on Automatic Control, 1679–1684.

[21] Moreira, M. V., Lesage, J.-J., 2018. Enhanced discrete event model for system identification with the aim of fault detection. In: 14th Workshop on Discrete Event Systems. IFAC, Sorrento Coast, Italy, pp. 172–178.

[22] Qiu, W., Kumar, R., 2006. Decentralized failure diagnosis of discrete event systems. IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans 36 (2), 384–395.

[23] Real Games, 2018. FACTORY I/O. URL `http://factoryio.com`

[24] Roth, M., 2010. Identification and fault diagnosis of industrial closed-loop discrete event systems. PhD dissertation, Ecole Normale Superieure de Cachan and Technische Universitat Kaiserlautern.

[25] Roth, M., Lesage, J.-J., Litz, L., 2009. An FDI method for manufacturing systems based on an identified model. In: 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM2009). Moscow, Russia, pp. 1406–1411.

[26] Roth, M., Lesage, J.-J., Litz, L., 2011. The concept of residuals for fault localization in discrete event systems. Control Engineering Practice 19 (9), 978–988.

[27] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D., 1995. Diagnosability of discrete-event systems. IEEE Trans. on Automatic Control 40 (9), 1555–1575.

[28] Santoro, L. P. M., Moreira, M. V., Basilio, J. C., 2017. Computation of minimal diagnosis bases of discrete-event systems using verifiers. Automatica 77, 93–102.