Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

COPPE UFRJ

TIMED AUTOMATON WITH OUTPUTS AND CONDITIONAL TRANSITIONS: MODELING, IDENTIFICATION, AND FAULT DETECTION

Ryan Pitanga Cleto de Souza

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Marcos Vicente de Brito Moreira

Rio de Janeiro
Setembro de 2020

TIMED AUTOMATON WITH OUTPUTS AND CONDITIONAL TRANSITIONS: MODELING, IDENTIFICATION, AND FAULT DETECTION

Ryan Pitanga Cleto de Souza

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Orientador: Marcos Vicente de Brito Moreira

Aprovada por: Prof. Marcos Vicente de Brito Moreira
Prof. João Carlos dos Santos Basilio
Prof. André Bittencourt Leal

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2020

Souza, Ryan Pitanga Cleto de

Timed automaton with outputs and conditional transitions: modeling, identification, and fault detection/Ryan Pitanga Cleto de Souza. – Rio de Janeiro: UFRJ/COPPE, 2020.

XV, 55 p.: il.; 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2020.

Referências Bibliográficas: p. 52 – 55.

1. Fault diagnosis. 2. System identification. 3. Discrete-event systems. 4. Finite automata. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Acknowledgments

Meus agradecimentos vão a todos os colegas com quem tive a oportunidade de conviver durante o mestrado, tanto durante as aulas quanto no laboratório. Certamente aprendi bastante graças a esse convívio.

Agradeço também ao meu orientador por todos os conselhos que contribuíram significativamente para a qualidade do trabalho realizado, não só nesta dissertação mas também nos artigos que foram publicados.

Também agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro durante a realização do mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AUTÔMATO TEMPORIZADO COM SAÍDAS E TRANSIÇÕES CONDICIONAIS: MODELAGEM, IDENTIFICAÇÃO E DETECÇÃO DE FALHAS

Ryan Pitanga Cleto de Souza

Setembro/2020

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

Neste trabalho, um método para detecção de falhas de Sistemas a Eventos Discretos (SED) baseado em um modelo temporizado chamado Autômato Temporizado com Saídas e Transições Condicionais (ATSTC), obtido por identificação, é apresentado. O ATSTC é uma extensão de um modelo não-temporizado recentemente proposto na literatura, chamado Autômato Determinístico com Saídas e Transições Condicionais (ADSTC). Diferentemente do ADSTC, no qual apenas o comportamento lógico do SED é considerado, o ATSTC leva em consideração informação sobre o tempo em que os eventos são observados, e, por esse motivo, pode ser utilizado para a detecção de falhas que não podem ser detectadas através do uso de modelos não-temporizados, como falhas que levam o detector de falhas a estados de bloqueio. O ATSTC representa o comportamento livre de falhas do sistema, e uma falha é detectada quando o comportamento observado é diferente daquele previsto pelo modelo, considerando tanto informação lógica quanto temporal. Além do procedimento de identificação necessário à obtenção do modelo ATSTC, o algoritmo para detecção de falhas também é apresentado. Um exemplo prático é utilizado para ilustrar os resultados do método proposto.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TIMED AUTOMATON WITH OUTPUTS AND CONDITIONAL TRANSITIONS: MODELING, IDENTIFICATION, AND FAULT DETECTION

Ryan Pitanga Cleto de Souza

September/2020

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

In this work, a method for fault detection of Discrete-Event Systems (DES) based on a timed model called Timed Automaton with Outputs and Conditional Transitions (TAOCT), obtained by identification, is presented. The TAOCT is an extension of an untimed model recently proposed in the literature, called Deterministic Automaton with Outputs and Conditional Transitions (DAOCT). Differently from the DAOCT, where only the logical behavior of the DES is considered, the TAOCT takes into account information about the time that the events are observed, and, for this reason, it can be used for the detection of faults that cannot be detected by using untimed models, such as faults that lead the fault detector to deadlocks. The TAOCT represents the fault-free system behavior, and a fault is detected when the observed behavior is different from the one predicted by the model, considering both logical and timing information. In addition to the identification procedure required to obtain the TAOCT model, the algorithm for fault detection is also presented. A practical example is used to illustrate the results of the proposed method.

Contents

List of Figures

List of Tables

List of Symbols

$\mathbb{R}$	The set of real values, p. 6
Σ	Set of events, p. 7
ε	Empty sequence, p. 7
$\lVert . \rVert$	Length of an (untimed or timed) event sequence, p. 7
$\Sigma^\star$	Kleene-closure of set Σ, p. 8
$\overline{L}$	Prefix-closure of language L, p. 8
G	Automaton, p. 9
X	Set of states, p. 9
f	Transition function, p. 9
x_0	Initial state, p. 9
X_m	Set of marked states, p. 9
Γ	Active event function, p. 9
2^A	Power set of A, *i.e.*, the set formed of all subsets of A, p. 9
$!$	'is defined', p. 9
$L(G)$	Language generated by automaton G, p. 10
c_g	Global clock, p. 11
$c_g(t)$	Global clock value at time instant $t \in \mathbb{R}^+$, p. 11
$\mathbb{R}^+$	The set of non-negative real values, p. 11
$guard$	Guard function of the TATI model, p. 11
$\mathcal{A}$	Set of admissible timing intervals, p. 11

Σ_t	Set of all possible timed events, p. 12
$\Sigma_t^\star$	Set of all sequences formed of elements $\sigma_t \in \Sigma_t$, including the empty sequence ε, p. 12
$\overline{L_t}$	Prefix-closure of timed language L_t, p. 12
m_i	Number of binary input signals, p. 12
i_h	h-th input signal, p. 12
m_o	Number of binary output signals, p. 12
o_h	h-th output signal, p. 12
$u(t)$	I/O vector at time t, p. 12
$\mathbb{N}$	The set of non-negative integers, p. 13
p_{u_i}	i-th observed untimed path, p. 13
l'_i	Length of untimed path p_{u_i}, p. 13
r	Number of observed untimed paths, p. 13
ψ	Function that extracts the sequence of events from an untimed path, p. 13
P_u	Set of observed untimed paths, p. 13
L_{Obs}	Observed language, p. 14
L_{Orig}	Original fault-free language, p. 14
L_{Iden}	Language generated by the identified model, p. 14
L_{Exc}	Exceeding language generated by the model, p. 14
L_{OrigNI}	Unidentified language of the system, p. 15
$L_{Orig}^{\leq n_0}$	Set containing all sequences in L_{Orig} with length up to n_0, p. 15
$\setminus$	Set difference, p. 15
$L_{Obs}^{\leq n_0}$	Set containing all sequences in L_{Obs} with length up to n_0, p. 15
$\emptyset$	Empty set, p. 15

$L_{OrigNI}^{\leq n_0}$	Set containing all sequences in L_{OrigNI} with length up to n_0, p. 15
$Sub(s)$	Set of subsequences of sequence s, p. 15
$L_{S,Obs}^{n}$	Language formed of all observed subsequences of events of length n, p. 15
$L_{S,Iden}^{n}$	Language formed of all subsequences of events of the identified model of length n, p. 15
k	Free parameter for parametric identification, p. 16
$p_{u_i}^{k}$	i-th modified path, p. 16
$\uparrow d$	Rising edge of the d-th controller signal, p. 16
$\downarrow d$	Falling edge of the d-th controller signal, p. 16
Ω	Set of observed I/O vectors, p. 17
λ	State output function, p. 17
R	Set of path indexes, p. 17
θ_u	Path estimation function of the DAOCT, p. 17
X_f	Set of final states, p. 17
$\mathbb{N}_1$	Set $\{0,1\}$, p. 17
$\theta_{u,e}$	Extended path estimation function of the DAOCT, p. 18
$L(\text{DAOCT})$	Language generated by the DAOCT model, p. 18
$\lvert.\rvert$	Set cardinality, p. 20
ν	Number of observed timed paths, p. 21
p_q	q-th observed timed path, p. 22
l_q	Length of the q-th observed timed path, p. 22
s_t^q	Timed sequence associated with path p_q, p. 22
$L_{t,Obs}$	Observed timed language, p. 22
P	Set of observed timed paths, p. 22

P_i	Subset of P containing all timed paths associated with untimed path p_{u_i}, p. 22
p'_i	i-th time-interval path, p. 22
P'	Set of time-interval paths, p. 22
g	Guard function of the TAOCT model, p. 23
$\mathcal{C}$	Set of admissible constraints, p. 23
$\phi(s_t)$	Event sequence associated with timed sequence s_t, p. 23
θ	Path estimation function of the TAOCT, p. 23
θ_e	Extended path estimation function of the TAOCT, p. 24
$L_{t,Iden}$	Timed language generated by the TAOCT model, p. 24
ζ	User-specified threshold of the Nearest Neighbors Method, p. 25
δ	Uncertainty in the measurement of event timings, p. 26
d_K	Total distortion with K clusters, p. 29
K_{max}	User-specified maximum number of clusters, p. 29
$\mathcal{E}$	Evaluating function, p. 29
ξ	Free parameter for time-interval formation based on K-Means, p. 30
p'^k_i	i-th timed modified path, p. 32
P'^k	Set of timed modified paths, p. 33
$\tilde{\lambda}$	Function that associates a state to an element in Ω^k, p. 34

List of Abbreviations

DES	Discrete-Event Systems, p. 1
NDAAO	Non-Deterministic Autonomous Automaton with Outputs, p. 2
PLC	Programmable Logic Controller, p. 2
DAOCT	Deterministic Automaton with Outputs and Conditional Transitions, p. 2
TAAO	Timed Autonomous Automaton with Outputs, p. 4
TAOCT	Timed Automaton with Outputs and Conditional Transitions, p. 4
LCA	Laboratory of Control and Automation, p. 5
UFRJ	Federal University of Rio de Janeiro, p. 5
TATI	Timed Automaton with Timing Intervals, p. 11

Chapter 1

Introduction

In the past years, interest in the domain of fault diagnosis of Discrete-Event Systems (DES) has increased. Since the introduction of the concept of fault diagnosis and diagnosability analysis of DES in SAMPATH *et al.* [1], several methods have been proposed in the literature for fault diagnosis of untimed DES [2–6], and of timed DES [7–11]. These methods provide a theoretical framework for the study of fault diagnosis of DES. However, their application to complex real systems, such as industrial plants, is a difficult task since these methods rely on an accurate model of the system, including its post-fault behavior.

In general, the modeling of a real-world system is very laborious and time consuming, since, depending on its size and complexity, it is very difficult to take into account all possible behaviors of the system in the model. This problem is amplified when the post-fault behavior is considered, since there may exist unpredictable consequences to a fault occurrence. In addition, the modeling process requires engineers that know the plant behavior, and are familiar with discrete-event modeling techniques. All these problems restrict the application of methods based on the complete system model to small systems, where the fault-free and faulty behaviors can be completely known.

In order to overcome the modeling difficulties that arise due to the aforementioned problems, some solutions based on system identification were presented using Petri net models [12–18]. However, most of these works do not address the problem of fault diagnosis. An identification method suitable for fault diagnosis is presented in CABASINO *et al.* [17], but only the faulty behavior is obtained by identification, and complete knowledge of the fault-free behavior of the system is required. In DOTOLI *et al.* [14], even though complete knowledge of the fault-free behavior of the system is not required, partial knowledge about the model is assumed, *e.g.*, it is necessary to provide an upper bound on the number of places in the net. In DOTOLI *et al.* [16], knowledge about the faulty behavior is not required, and the problem of identifying the unobservable behavior of a DES is addressed. However, in DOTOLI *et al.* [16], the fault-free system structure and dynamics are assumed to be known.

In the Petri net identification methods proposed in the literature with the objective of fault detection, it is supposed that the system structure or dynamics are completely or partially known, which makes this formalism suitable for modeling these systems. However, when no previous knowledge of the system is given, then automata become a suitable formalism for identification due to its more basic structure. In addition, generating an automatic fault detection method for systems modeled by automata is simpler than for systems modeled by Petri nets. Since, in this work, we assume that no previous knowledge about the system is available, then we have focused on the identification of automaton models.

Fault detection techniques based on an identified automaton model of the system have been proposed in the literature [19–22]. In these works, the two main ideas are: (i) 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 localization [22, 23].

A model for the identification of closed-loop industrial DES, called Non-Deterministic Autonomous Automaton with Outputs (NDAAO), is presented in KLEIN *et al.* [19]. The identification procedure is based on the acquisition of binary signals exchanged between the Programmable Logic Controller (PLC) and the plant. These signals correspond to sensor readings (inputs to the controller) and actuator commands (outputs of the controller), as shown in Figure 1.1. The NDAAO is identified from observed fault-free paths of the system, composed of sequences of vectors whose entries are the values of the binary input and output signals of the controller. These sequences form the input data of the identification procedure. With a view to obtaining a compact model, loops are introduced in the identified NDAAO, leading to the generation of sequences that have not been observed during the identification process. These sequences form the exceeding language generated by the identified model, and can be associated with non-detectable faults.

In MOREIRA and LESAGE [21], a different automaton model, called Deterministic Automaton with Outputs and Conditional Transitions (DAOCT), is proposed. Conditions for the transposition of the transitions associated with the observed fault-free paths used in the identification procedure are added to the model by using a path estimation function. The use of the path estimation function reduces the exceeding language generated by the model in comparison with the NDAAO, and, consequently, it reduces the number of non-detectable faults. In MOREIRA and LESAGE [22], an algorithm for fault diagnosis using the DAOCT model proposed in MOREIRA and LESAGE [21] is presented. In DE SOUZA *et al.* [24], a hierar-

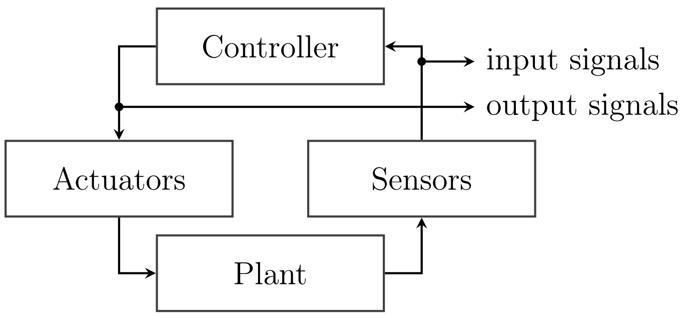


Figure 1.1: Closed-loop system showing the signals exchanged between plant and controller.

chical model using the DAOCT is introduced as a way of making it easier to obtain the fault-free model of the system, which is possible due to the path information embedded in the structure of the DAOCT model.

It is important to remark that some faults cannot be detected by using untimed models. For instance, faults that prevent the system from generating events, leading the fault detector to a deadlock, cannot be detected from the logical behavior of the system. In addition, in some cases, timing information of event occurrences can be used to distinguish behaviors of the system, improving the capability of detecting faults.

In the context of time Petri net identified models, some interesting results have been reported in the literature. In BASILE *et al.* [25], information about the timing of event occurrences is added as a way of speeding up the optimization method used for computing a time Petri net model for the system. This method is based on the solution of a mixed integer linear programming problem. The major drawback of the method presented in BASILE *et al.* [25] is that it requires the knowledge of the number of places and transitions in the net, as well as the timing for the firing of each transition. Thus, the timing behavior is not determined by identification, which prevents its application to complex industrial systems for which, in general, this kind of information is not available.

In BASILE *et al.* [26], it is assumed that a time Petri net model for the nominal behavior of the system is available, and models for the faulty behaviors are identified using the discrepancies between the observed and nominal behaviors of the system. In BASILE *et al.* [27], a time Petri net that models the complete system, and that can be used for fault detection, is identified. In BASILE *et al.* [26] and BASILE *et al.* [27], the adopted strategy for computing the identified model is based on the solution of a mixed integer linear programming problem, as in BASILE *et al.* [25].

A timed automaton obtained by identification is used in SUPAVATANAKUL *et al.* [28] in a fault diagnosis scheme, where the measured data on which the identification model is built is based on the quantization of continuous signals, and not on discrete binary signals. Fault isolation is addressed in SUPAVATANAKUL *et al.* [28] by computing identified models for the faulty behavior of the system, for every fault that must be diagnosed. In this case, fault isolation is restricted only to the previously considered faults. Another drawback of this approach is the fact that data for describing the system faulty behaviors is, in general, not available in practice.

The NDAAO model presented in KLEIN *et al.* [19] is extended in SCHNEIDER [29] to consider timing information, and timing constraints in the form of guards are added to each transition. Each transition has a guard that is defined as a single timing interval, and the guard pre-condition is satisfied when the event occurs in this interval. This model, called Timed Autonomous Automaton with Outputs (TAAO), is capable of detecting faults if the system reaches a deadlock. The determination of the time guards associated with each transition must be carried out in such a way that the total numbers of non-detectable faults and false alarms are minimized in a fault monitoring scheme. In DAS and HOLLOWAY [30], the accepted time delays associated with a transition are determined using observations of the fault-free system. However, this method assumes that the delays are normally distributed for a given transition, which restricts its application to systems where this hypothesis holds. The method for determining the time guards for the TAAO, on the other hand, is based on a method that does not rely on an a-priori known distribution and is suited for dealing with asymmetry in time values [31]. Both in DAS and HOLLOWAY [30] and SCHNEIDER *et al.* [31], the aim is to handle disturbances in timing behavior during accepted system operation (which may lead the fault detector to issue a false alarm), while ensuring that all faults are effectively detected. A trade-off between non-detectable faults and false alarms is then required so that fault detection can be properly performed.

In this work, a new timed model for DES identification with the aim of fault detection is presented. The new model, called Timed Automaton with Outputs and Conditional Transitions (TAOCT), is an extension of the DAOCT model in which timing information is added to the transitions in the form of guards for the transposition of the transitions. Since the TAOCT is based on the DAOCT, it inherits the same advantages of the underlying DAOCT model in terms of exceeding language reduction in comparison with the TAAO model, which is based on the NDAAO. In addition, differently from the TAAO model, the procedure to define the guards of the transitions of the TAOCT is capable of distinguishing different timing intervals associated with the same transition. More specifically, a guard can be defined either as a single timing interval or as a disjoint union of several timing intervals. Thus, more accurate guards can be defined, increasing the number of detectable faults in comparison with the TAAO model. In addition, the timing information can also be used to refine the path estimate carried out by the path estimation function of the underlying DAOCT, reducing the number of non-detectable faults. It is important to remark that, as in the TAAO, deadlocks caused by faults can also be detected by using the TAOCT.

Most of the results presented in this work are based on DE SOUZA *et al.* [32] and DE SOUZA *et al.* [33]. A formal introduction to the TAOCT is provided in both these works, and a more thorough discussion of the fault detection strategy is presented in DE SOUZA *et al.* [33].

This work is organized as follows. In Chapter 2, preliminary concepts are presented. In this chapter, a brief introduction to languages and automata is followed by the presentation of a timed model and some related concepts. The DAOCT is also presented in Chapter 2, given that the TAOCT and the procedure to obtain it are largely based on results obtained for the DAOCT model.

In Chapter 3, the TAOCT model is formally introduced and the identification procedure, with all the steps required to compute it, is presented. Some didactic examples are also shown in order to illustrate the modeling process.

In Chapter, 4, the scheme for fault detection using the TAOCT is introduced. The algorithm for performing fault detection is presented and the different faulty scenarios where the fault is detected using the TAOCT thanks to its timing structure are discussed.

In Chapter 5, a practical example is presented in order to illustrate the application of the proposed method to a real system: a didactic manufacturing plant of the Laboratory of Control and Automation (LCA), located at the Federal University of Rio de Janeiro (UFRJ). We describe in this chapter the procedure of computing the TAOCT model for this system and discuss some interesting results concerning fault detection.

The concluding remarks are presented in Chapter 6.

Chapter 2

Theoretical background

In this chapter, an overview of some useful concepts is presented. Sections 2.1, 2.2 and 2.3 provide an introduction to some basic topics in DES theory, and then in Section 2.4 these concepts are extended to a timed framework. This chapter ends in Section 2.5 with the presentation of the DAOCT model, on which the TAOCT is based.

2.1 Discrete-event systems

For continuous-variable systems, the evolution of the model can be expressed in the form of differential equations, in the case of continuous-time systems, or difference equations, in the case of discrete-time systems. The solutions to these equations allow to determine the state of the model at any given time instant, from any initial state. In the case of continuous systems, the state space is continuous. For example, it could be the set of real values $\mathbb{R}$. Since the transitions from one state to another are driven by the passage of time (both for continuous- and discrete-time systems), it is said that these systems are *time-driven* [34].

However, the formalism used for continuous-variable systems is not suited for the mathematical description of DES. In fact, in a discrete-event model, the state space is a discrete set, and the transitions between states of the model are instantaneous and are executed asynchronously in certain time instants. These transitions are triggered by event occurrences, which are responsible for the evolution of a discrete-event system. Thus, the dynamics of the system, instead of being driven by the passage of time, are driven by the occurrence of asynchronous events. For this reason, DES are said to be *event-driven*. For example, in the particular case of an industrial setup, a command being sent to an actuator or a system variable reaching a specific value can both be modeled as events, and upon their occurrence the system transitions instantaneously to another state belonging to the state space, remaining in the same state until the occurrence of another event at a later time instant.

Hence, DES satisfy the following two properties [34]:

1. The state space is a discrete set.

2. The state transition mechanism is event-driven.

According to CASSANDRAS and LAFORTUNE [34], DES may be informally defined in the following manner:

Definition 2.1 *A discrete-event system is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.* □

It often suffices to model a discrete-event system by the orderings of events that it can execute, which describe the *logical behavior* of the system. In Sections 2.2 and 2.3, two mathematical representations of the logical behavior are introduced: languages and automata.

However, in some cases, it may be interesting to take into account the timing behavior of the system in the model. Then, instead of simply considering orderings of events, the times at which the events take place must also be included. A simple timed model is introduced and briefly discussed in Section 2.4.

2.2 Languages

Let Σ denote the set of events of a discrete-event system. Sequences can be formed by selecting events from Σ. A sequence that has no events is called the empty sequence, denoted as ε. The length of a sequence of events s is the number of events contained in it, including multiple occurrences, and is denoted as $\|s\|$. By convention, the length of the empty sequence is equal to zero [34]. A language is then defined as follows [34].

Definition 2.2 *A language defined over an event set Σ is a set of finite-length sequences formed from events in Σ.* □

Examples of languages are given in the sequel [34].

Example 2.1 *Let $\Sigma = \{a, b, c\}$ be a set of events. Then, it is possible to define languages L_1, L_2 e L_3 in the following manner:*

$$\begin{aligned}
L_1 &= \{\varepsilon, a, abb\},\\
L_2 &= \{\textit{all sequences of length 3 starting with event } a\},\\
L_3 &= \{\textit{all sequences of finite length starting with event } a\}.
\end{aligned}$$

□

The operation that allows the construction of sequences from events of a set Σ is called *concatenation*. The concatenation of two sequences u and v forms a new sequence uv with the events in u followed by the events in v in the same order. Empty sequence ε is the neutral element of concatenation: $u\varepsilon = \varepsilon u = u$ for any sequence u.

Let $\Sigma^\star$ be the set formed of all possible finite sequences constructed with events from Σ, including ε. This operation is called *Kleene-closure* [34]. For example, if $\Sigma = \{a, b, c\}$, then we have that

$$\Sigma^\star = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, ...\}.$$

Note that all languages that can be formed with the elements of Σ are subsets of $\Sigma^\star$.

Consider now a sequence s that can be written as $s = tuv$. The following definitions can be made:

- t is called a *prefix* of s,
- u is called a *subsequence* of s,
- v is called a *suffix* of s.

Since languages are subsets of $\Sigma^\star$, some operations from set theory, such as union, intersection, difference and complement with respect to $\Sigma^\star$ are also valid for languages. In addition to all these operations, several other operations specific to languages can also be defined [34]. One of these operations, called *prefix-closure*, is defined as follows.

Definition 2.3 *Let $L \subseteq \Sigma^*$. Then, the prefix-closure of L, denoted as $\overline{L}$, is given by:*

$$\overline{L} := \{s \in \Sigma^\star : (\exists t \in \Sigma^\star)[st \in L]\}.$$

□

The prefix-closure of L contains all prefixes of all sequences of L. Language L is said to be prefix-closed if $L = \overline{L}$, *i.e.*, if every prefix of a sequence in L also belongs to L.

An example of prefix-closure is given as follows [34].

Example 2.2 *Let $\Sigma = \{a, b, c\}$ and consider two languages $L_1 = \{\varepsilon, a, abb\}$ and $L_2 = \{c\}$. Then, the prefix-closure of L_1 and L_2 are given by $\overline{L_1} = \{\varepsilon, a, ab, abb\}$ and $\overline{L_2} = \{\varepsilon, c\}$, respectively.* □

2.3 Automata

Languages describe the behavior of a discrete-event system by listing all possible sequences of events generated by the system. However, it is not always convenient to work with this way of modeling system behavior, and thus more appropriate formalisms, with more compact structures, must be used in order to model the behavior of DES more conveniently.

In this section, a common and particularly useful formalism, known as *automaton*, is presented. The main concepts related to automata are briefly discussed and it will be seen that they provide a compact description of the logical behavior of DES [34]. Firstly, we present the formal definition of a deterministic automaton [34].

Definition 2.4 *A deterministic automaton, denoted as G, is a 5-tuple*

$$G = (X, \Sigma, f, x_0, X_m),$$

where:

- *X is the set of states;*
- *Σ is the finite set of events;*
- *$f : X \times \Sigma \to X$ is the transition function;*
- *x_0 is the initial state.*
- *$X_m \subseteq X$ is the set of marked states.* □

With respect to the transition function, $f(x, \sigma) = y$ means that there is a transition labeled by event $\sigma \in \Sigma$ from state $x \in X$ to state $y \in X$. Such transition is represented by the triple (x, σ, y). We consider that function f may be partially defined on its domain.

Active event function $\Gamma : X \to 2^{\Sigma}$ is defined as follows: $\Gamma(x) := \{\sigma \in \Sigma : f(x, \sigma)!\}$, where the symbol '!' denotes 'is defined'. In words, $\Gamma(x)$ returns the set of events that label some transition from state x in the automaton. These events are said to be *feasible* at state x.

The set of marked states X_m is usually formed of the states of the automaton that have some special meaning, which may correspond, for instance, to the successful completion of a given task.

Initially, automaton G is at state x_0. Upon the occurrence of some event $\sigma \in \Gamma(x_0)$, the model transitions to the state given by $f(x_0, \sigma)$. This process then

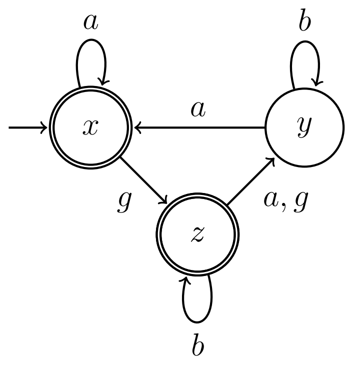


Figure 2.1: State transition diagram for the automaton of Example 2.3.

continues according to the events that are feasible at each state in the model. Consider the following example, where a graphical representation of an automaton is shown in the form of a state transition diagram [34].

Example 2.3 *Consider the state transition diagram represented in Figure 2.1. In this case, $X = \{x, y, z\}$, $\Sigma = \{a, b, g\}$, initial state x_0 is state x and $X_m = \{x, z\}$. The states given by the transition function f for each element in $X \times \Sigma$ where f is defined are given by: $f(x, a) = x$, $f(x, g) = z$, $f(y, a) = x$, $f(y, b) = y$, $f(z, a) = y$, $f(z, b) = z$ and $f(z, g) = y$.* □

The domain of the transition function can be extended in order to consider sequences of events instead of single events [34]. The extended transition function $f : X \times \Sigma^\star \to X$ is recursively defined as follows.

$$\begin{aligned} f(x, \varepsilon) &:= x \\ f(x, s\sigma) &:= f(f(x, s), \sigma) \text{ for } s \in \Sigma^\star \text{ and } \sigma \in \Sigma. \end{aligned}$$

For the rest of this work, only the extended version of the transition function is considered.

The language generated by an automaton $G = (X, \Sigma, f, x_0, X_m)$ is defined as follows [34].

$$L(G) := \{s \in \Sigma^\star : f(x_0, s)!\}.$$

Thus, the language generated by an automaton is the set containing all possible sequences of events that may occur in the automaton starting at its initial state. An example illustrating the concept of generated language is presented in Example 2.4.

Example 2.4 *Consider the automaton G represented in Figure 2.2. In this case, we have that $X = \{0, 1\}$ and $\Sigma = \{a, b\}$. Note that all events in Σ are feasible in any state of G. Thus, from the initial state, any sequence formed with elements in Σ can be generated. Therefore, $L(G) = \Sigma^\star$.* □

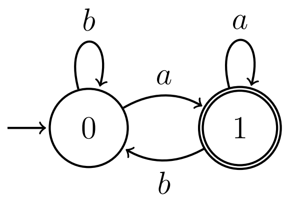


Figure 2.2: State transition diagram of the automaton of Example 2.4.

2.4 Timed Automaton with Timing Intervals

Even though the deterministic automaton presented in Definition 2.4 is capable of modeling the behavior of DES in terms of the generated sequences of events, in many cases it is also necessary to consider the time in which the events take place. With this in mind, a timed DES model is formally introduced in this section. This model is called Timed Automaton with Timing Intervals (TATI) and it is defined as follows [34].

Definition 2.5 (Timed Automaton with Timing Intervals) *A timed automaton with timing intervals is the six-tuple*

$$TATI = (X, \Sigma, f, c_g, guard, x_0),$$

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;*
- *c_g is the global clock with value $c_g(t) \in \mathbb{R}^+$, $t \in \mathbb{R}^+$;*
- *$guard : X \times \Sigma \to \mathcal{A}$ is the guard function, where $\mathcal{A}$ is the set of admissible timing intervals for the global clock c_g;*
- *$x_0 \in X$ is the initial state of the system.* □

Note that, differently from Definition 2.4, the set of marked states is not defined for the TATI. The reason for this is simply that, in this work, marked states are not relevant in this timed model. Therefore, they are not included in the definition of the TATI.

A timed automaton with timing intervals is an automaton to which a global clock c_g is added. Thus, timed sequences are generated by the TATI. Function $guard : X \times \Sigma \to \mathcal{A}$ specifies the timing conditions that need to be satisfied on the global clock for the transition to occur, and $\mathcal{A}$ are the admissible clock constraints defined as timing intervals. It is important to remark that $guard(x, \sigma)$ is defined if, and only if, $f(x, \sigma)$ is defined. In the TATI, the global clock is reset to zero each time an event occurs. Let t' denote the time that a state $x \in X$ is reached in the timed automaton, and let τ denote the time that has elapsed after reaching state x. Then, transition (x, σ, x'), where $x' = f(x, \sigma)$, is executed in the TATI if event σ occurs at time $t' + \tau$, and $\tau \in guard(x, \sigma)$.

The TATI is a simplified version of the timed automaton with guards [35]. Later on, a similar formalism is used to add timing information to the proposed identification model.

A timed event σ_t is a pair (σ, τ), with $\sigma \in \Sigma$ and $\tau \in \mathbb{R}^+$. The set of all possible timed events, considering the events in Σ, is given by $\Sigma_t := \Sigma \times \mathbb{R}^+$. The set of all sequences formed of elements $\sigma_t \in \Sigma_t$, including the empty sequence ε, is denoted by $\Sigma_t^\star$. Note that sequences of timed events are concatenated exactly in the same way described in Section 2.2 for untimed events.

In general, in the literature, timed sequences consider an absolute time that specifies when the event has occurred with respect to the beginning of the sequence execution, and not with respect to the sojourn time of each state of the path. However, the same information is contained in both types of timed sequences, since it is always possible to obtain a timed sequence with respect to the beginning of the process, from the timed sequence considering time durations s_t, and vice-versa. In this work, the latter approach is adopted. Hence, a timed sequence $(\sigma_1, \tau_1)(\sigma_2, \tau_2) \ldots (\sigma_l, \tau_l) \in \Sigma_t^\star$ is said to be generated by the timed automaton if there exist states $x_1, x_2, \ldots, x_l \in X$ such that $x_{i+1} = f(x_i, \sigma_{i+1})$ and $\tau_{i+1} \in guard(x_i, \sigma_{i+1})$, for $i = 0, \ldots, l-1$.

Similarly to the concept of prefix-closure presented for untimed languages in Definition 2.3, given a timed language $L_t \subseteq \Sigma_t^\star$, the prefix-closure of L_t is given by $\overline{L}_t = \{w_t \in \Sigma_t^\star : (\exists z_t \in \Sigma_t^\star)[w_t z_t \in L_t]\}$.

2.5 Deterministic Automaton with Outputs and Conditional Transitions

Let us consider the closed-loop system depicted in Figure 1.1, 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

$$u(t_1) = \begin{bmatrix} i_1(t_1) & \ldots & i_{m_i}(t_1) & o_1(t_1) & \ldots & o_{m_o}(t_1) \end{bmatrix}^T$$

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. This leads to the following definition.

Definition 2.6 *An event of the identified model is any observed instantaneous change in one or more signals of the I/O vector u.* □

Note that in Definition 2.6, it is possible for an event to be defined as the change of more than one I/O signal, since multiple signals may change their values during the same scan cycle of the PLC.

The I/O vector of the system $u(t_j)$, $j \in \mathbb{N}$, is simply represented 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), with σ denoting the event associated with the changes in signals from u_1 to 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 $p_u = (u_1, \sigma_1, u_2, \sigma_2, \ldots, \sigma_{l-1}, u_l)$[1].

Let us consider that the observed paths of the system are denoted as $p_{u_i} = (u_{i,1}, \sigma_{i,1}, u_{i,2}, \sigma_{i,2}, \ldots, \sigma_{i,l'_i-1}, u_{i,l'_i})$, for $i = 1, \ldots, r$, where r is the number of observed paths, and l'_i is the number of vertexes of each path p_{u_i}. Associated with each path p_{u_i} there is a sequence $s_i = \psi(p_{u_i}) = \sigma_{i,1}\sigma_{i,2}\ldots\sigma_{i,l'_i-1}$, where $\psi : P_u \to \Sigma^\star$ with $P_u = \{p_{u_1}, \ldots, p_{u_r}\}$. The following assumptions regarding the observed paths are considered in KLEIN *et al.* [19], ROTH *et al.* [20], MOREIRA and LESAGE [21] and MOREIRA and LESAGE [22].

A1. The system has a unique initial state, whose corresponding I/O vector is denoted by u_0, and all observed paths start at the initial state of the system, with the same I/O vector u_0.

A2. The complete sequence of events associated with a path p_{u_i} is not a prefix of the sequence of events of another path p_{u_z}, $\forall i, z \in \{1, 2, \ldots, r\}$ such that $i \neq z$.

Assumption **A1** states that the first I/O vector is the same for all observed paths. This I/O vector may correspond to the beginning of a production cycle, in the case of an industrial system.

It is important to remark that Assumption **A2** holds true in several cases, since each observed path is associated with a system task, and, in a large number of applications, the execution of a sequence of events associated with a system task is not a prefix of another sequence of events associated with other possible task executed by the system.

[1]The 'u' in p_u refers to 'untimed', so that the notation for these paths is distinct from the one used for the timed paths introduced in Chapter 3.

The following definition of the language observed by the system can be stated[21]:

$$L_{Obs} := \bigcup_{i=1}^{r} \overline{\{s_i\}}. \tag{2.1}$$

Let L_{Orig} denote the unknown original fault-free language generated by the system. It is usually not possible to observe, in finite time, all sequences that belong to L_{Orig}. Thus, it can only be assured that $L_{Obs} \subseteq L_{Orig}$.

The objective of system identification is to find a model that simulates the observed fault-free behavior described by L_{Obs}. Thus, the language generated by the identified model, L_{Iden}, must satisfy $L_{Obs} \subseteq L_{Iden}$. In other words, the behavior of the system during identification (L_{Obs}) must be a part of the identified behavior (represented by L_{Iden}), in which case it is said that the identified model simulates the observed system behavior.

After obtaining the identified model that simulates the fault-free behavior, it can be used for fault detection by comparing the observed events generated by the closed-loop system with the behavior of the identified model. If the observed behavior is different from the predicted behavior, the fault is detected.

Two other languages of interest can be defined [21]:

- $L_{Exc} := L_{Iden} \backslash L_{Orig}$;
- $L_{OrigNI} := L_{Orig} \backslash L_{Iden}$.

As previously stated, only a part of the language generated by the identified model L_{Iden} actually corresponds to the observed fault-free behavior of the system. The rest of language L_{Iden} may contain sequences that belong to the language generated by the system L_{Orig} even though they have not been observed. The sequences that are generated by the identified model and that are not part of the fault-free behavior of the system form the exceeding language L_{Exc}. These sequences belonging to L_{Exc} arise due to loops introduced in the identified automaton model.

Since the fault detection strategy relies on the comparison between the sequences that are observed from the system and those that are generated by the model, if a fault leads to an observed sequence that does not belong to L_{Orig} but is generated by the model, then the fault is not detected. Thus, L_{Exc} is formed of faulty sequences that cannot be detected. Language L_{OrigNI}, on the other hand, is formed of the sequences that are part of the fault-free behavior of the system but are not generated by the obtained model. It can be seen that the sequences in L_{OrigNI} are interpreted by the fault detector as being faulty, even though they are actually part of the original fault-free behavior. In this case, a false alarm is raised by the detector. In order to ensure that fault detection is properly performed, both L_{Exc} and L_{OrigNI}

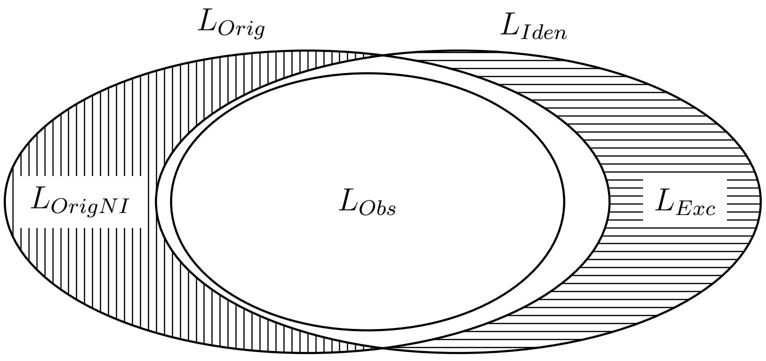


Figure 2.3: Diagram showing the relations between sets L_{Obs}, L_{Orig}, L_{Iden}, L_{OrigNI} and L_{Exc}.

must be reduced. Figure 2.3 shows the relations between L_{Obs}, L_{Orig}, L_{Iden}, L_{OrigNI} and L_{Exc}.

As shown in KLEIN *et al.* [19], if a sufficiently large number of controller vectors are observed, then there exists a number $n_0 \in \mathbb{N}$ such that $L^{\leq n_0}_{Orig} \backslash L^{\leq n_0}_{Obs} \approx \emptyset$, where $L^{\leq n_0}_{Orig}$ (resp. $L^{\leq n_0}_{Obs}$) denotes the subset of L_{Orig} (resp. L_{Obs}) formed of all paths with length less than or equal to n_0. Since $L_{Obs} \subseteq L_{Orig}$ and $L_{Obs} \subseteq L_{Iden}$, then $L^{\leq n_0}_{Orig} \backslash L^{\leq n_0}_{Obs} \approx \emptyset$ implies that $L^{\leq n_0}_{OrigNI} \approx \emptyset$, where $L^{\leq n_0}_{OrigNI}$ is the subset of L_{OrigNI} formed of all sequences of length up to n_0. In order to reduce the occurrence of false alarms, it is assumed in this work that $L^{\leq n_0}_{OrigNI} = \emptyset$. This implies that all sequences of length up to n_0 have been observed, or, equivalently, that all paths of length less than or equal to $n_0 + 1$ have been observed during the identification procedure. It is important to remark that, even if this assumption does not hold, fault detection can still be performed, but with a higher risk of raising false alarms.

In the sequel, the definition of a property called k-completeness is presented [21]. Before stating this definition, let us define function $Sub : \Sigma^\star \to 2^{\Sigma^\star}$ as $Sub(s) := \{w \in \Sigma^\star : (\exists t, v \in \Sigma^\star)[s = twv]\}$. Function $Sub(s)$ returns the set of subsequences of sequence s. Let us also define the language formed of all observed subsequences of events of length n as

$$L^n_{S,Obs} := \{s \in \Sigma^\star : (\|s\| = n)[\exists i \in \{1, 2, \ldots, r\}, s \in Sub(\psi(p_{u_i}))]\}.$$

The definition of k-completeness is given as follows [21].

Definition 2.7 *A model is said to be k-complete if for all $n \leq k$, $L^n_{S,Obs} = L^n_{S,Iden}$, where $L^n_{S,Iden}$ is the set formed of all subsequences of events of the identified model of length n.* □

In MOREIRA and LESAGE [21], an automaton model suitable for fault detection, called Deterministic Automaton with Outputs and Conditional Transitions (DAOCT), is proposed. The DAOCT is obtained from the observed paths p_{u_i},

$i = 1, \dots, r$, and a free parameter k. In order to do so, modified paths $p^k_{u_i}$ are computed from paths p_{u_i} such that the vertexes of $p^k_{u_i}$ are sequences of I/O vectors of length at most equal to k. The expression for $p^k_{u_i}$ is given by:

$$p^k_{u_i} = (y_{i,1}, \sigma_{i,1}, y_{i,2}, \sigma_{i,2}, \dots, \sigma_{i,l'_{i-1}}, y_{i,l'_i}), \tag{2.2}$$

where

$$y_{i,j} = \begin{cases} (u_{i,j-k+1}, \dots, u_{i,j}), & \textit{if } k \leq j \leq l'_i \\ (u_{i,1}, \dots, u_{i,j}), & \textit{if } j < k \end{cases}. \tag{2.3}$$

Note that the sequence of events of $p^k_{u_i}$ is equal to the sequence of events of path p_{u_i}. Thus, the only difference between p_{u_i} and $p^k_{u_i}$ is that each vertex of $p^k_{u_i}$ is now associated with a sequence of vectors instead of a single I/O vector , if $k > 1$. The role of the parameter k is to memorize in each vertex of path $p^k_{u_i}$ the occurrence of the last sequence of I/O vectors of length k. It can be seen that each vertex also stores the last $k-1$ events, as stated in the following lemma, presented in MOREIRA and LESAGE [21].

Lemma 2.1 *Each vertex $y_{i,j}$ of path $p^k_{u_i}$ stores the last $k-1$ events occurred in path $p^k_{u_i}$, if $j \geq k$, and the last $j-1$ events, if $j < k$.* □

It is important to remark that, if $k = 1$, then $p^k_{u_i} = p_{u_i}$, $i = 1, \dots, r$. The computation of modified paths is illustrated in the following example.

Example 2.5 *Consider a system with three controller signals and assume that the following paths have been observed during the identification procedure:*

$$p_{u_1} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, c, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, d, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, e, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$p_{u_2} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, g, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, h, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, c, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, d, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, j, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, l, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$p_{u_3} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, g, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, h, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, i, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, m, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, d, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, e, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)$$

Each event corresponds to different combinations of rising or falling edges of controller signals. The rising and falling edge of the d-th element of the I/O vector are denoted as $\uparrow d$ and $\downarrow d$, respectively. Thus, for instance, $a = \uparrow 2$ and $h = \uparrow 1.\uparrow 2$. By *choosing $k = 2$, the following modified paths are computed:*

$$p^2_{u_1} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, b, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, c, \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, d, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, e, \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \right),$$

$$p^2_{u_2} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, g, \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, h, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, b, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, c, \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, d, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \right.$$
$$\left. j, \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, l, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \right),$$

$$p^2_{u_3} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, g, \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, h, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, b, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, i, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, m, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \right.$$
$$\left. d, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, e, \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \right).$$

□

The formal definition of a DAOCT is stated in the sequel [21].

Definition 2.8 *A Deterministic Automaton with Outputs and Conditional Transitions (DAOCT) is the nine-tuple:*

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

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 indexes, $\theta_u : X \times \Sigma \to 2^R$ is the path estimation function, x_0 is the initial state, and X_f is the set of final states. □

Function λ associates each state $x \in X$ with output $\lambda(x)$, corresponding to the last recorded I/O vector. This is why "Automaton with Outputs" is part of the name of the DAOCT model. Since the transition function f is deterministic (an event can only lead to one state) and there is only one initial state x_0, the DAOCT model is deterministic. The path estimation function θ_u introduces additional conditions for the transposition of a transition, justifying the name "Conditional Transitions".

Each transition $x' = f(x, \sigma)$ of automaton DAOCT has a corresponding set $\theta_u(x, \sigma)$ of indexes that is associated with the paths p_{u_i} that contain transition (x, σ, x'). Function θ_u 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 this definition, it is first necessary to extend the domain of function θ_u to consider the execution of sequences of events, obtaining the extended path estimation function $\theta_{u,e} : X \times \Sigma^\star \to 2^R$, defined recursively as:

$$\theta_{u,e}(x, \varepsilon) = R,$$
$$\theta_{u,e}(x, s\sigma) = \begin{cases} \theta_{u,e}(x, s) \cap \theta_u(x', \sigma), \text{ where } x' = f(x, s), \text{if } f(x, s\sigma)! \\ \text{undefined, otherwise.} \end{cases}$$

The language generated by the DAOCT is given by

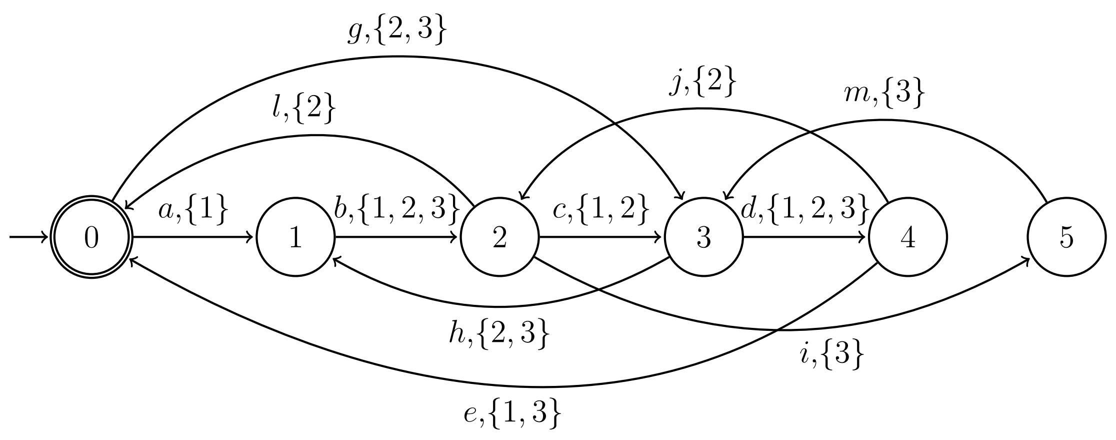
$$L(\text{DAOCT}) := \{s \in \Sigma^\star : f(x_0, s)! \wedge \theta_{u,e}(x_0, s) \neq \emptyset\}. \tag{2.4}$$

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 $\theta_{u,e}(x_0, s) \neq \emptyset$.

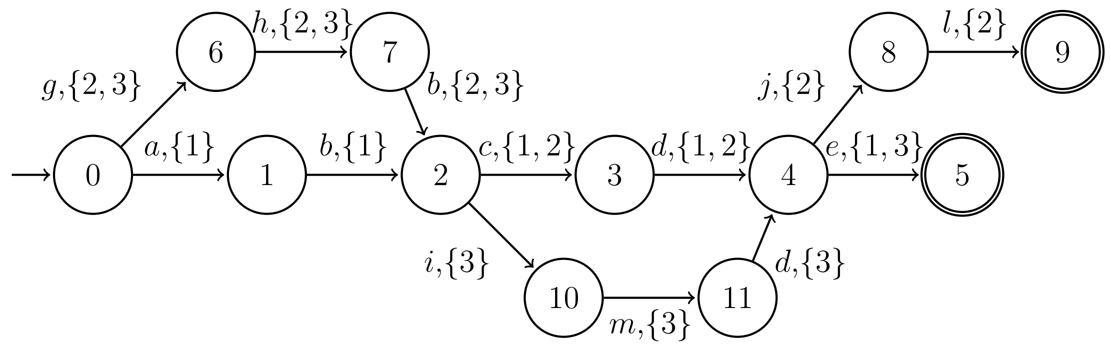
In MOREIRA and LESAGE [21], an algorithm for computing a DAOCT model from the modified paths $p^k_{u_i}$, $i = 1, \ldots, r$, is introduced. In the sequel, the DAOCT models obtained from the observed paths in Example 2.5 by using this algorithm are shown.

Example 2.6 *Consider the observed paths from Example 2.5. After applying the DAOCT identification algorithm [21], for $k = 1$ and $k = 2$, the automata shown in Figure 2.4 are obtained. The double circles denote the states that belong to set X_f. Each transition (x, σ, x') is labeled by its corresponding event σ and by the path estimation $\theta_u(x, \sigma)$.* □

By analyzing Figure 2.4, it is easy to see the impact of the choice of the parameter k on the resulting model. Indeed, for the case where $k = 2$, apart from the initial state, each state in the automaton is associated both with the last observed I/O vector and the one that preceded it. Consequently, the obtained model has more states in comparison with the case where $k = 1$, where each state is associated with a single I/O vector. As shown in Figure 2.4a, the DAOCT obtained with $k = 1$ has 6 states, which is the same number of different I/O vectors that were observed. On the other hand, the DAOCT obtained with $k = 2$ contains as many states as there are different vertexes in the modified paths $p^2_{u_i}$, $i = 1, 2, 3$ (see Example 2.5).



(a) $k = 1$.



(b) $k = 2$.

Figure 2.4: DAOCT models obtained in Example 2.6 for $k = 1$ and $k = 2$.

Note that, for $k = 1$, even though there are less states, some cycles are present in the automaton. As a result, the language generated by the model contains sequences that have not been observed and thus are not part of the fault-free system behavior. Therefore, these cycles increase the size of the exceeding language. It is possible to see how this happens by analyzing the automaton shown in Figure 2.4a. Note that the sequence *ghbl* can be executed by the automaton even without being recorded in any observed path. Hence, this sequence is associated with a fault that would not be detected in a fault detection scheme using this DAOCT model. However, by choosing $k = 2$, the resulting automaton does not have any cycles, as it can be seen in Figure 2.4b. In this case, the sequence *ghbl* can no longer be executed by the model, and thus a fault causing the occurrence of this sequence would be detected in a fault detection scheme using this model.

In MOREIRA and LESAGE [21], some important properties of the DAOCT model are presented and proven. These properties are listed as follows.

P1. $L_{Obs} \subseteq L(\text{DAOCT})$;

P2. For a given value of k, the identified DAOCT is k-complete, *i.e.*, $L_S^n(\text{DAOCT}) = L_{S,Obs}^n$, for all $n \leq k$;

P3. If the identified DAOCT does not have cyclic paths for a given value of k, then $L_{Exc} = \emptyset$.

Property **P1** states that the observed language L_{Obs} is a subset of the identified language generated by the DAOCT. In other words, the DAOCT model simulates the observed fault-free behavior of the system. Properties **P1** and **P2** show that the DAOCT model is suitable for fault detection, since, in addition to simulating the observed language, any subsequence of length k belongs to the DAOCT model if, and only if, it has been observed. This means that the approximation of the observed language given by the identified model can be made more accurate by increasing the value of k.

By increasing k, the number of cycles is reduced as a consequence of the higher number of states that are created (see Example 2.6). If the value of k is increased to the point where the DAOCT model becomes acyclic, then, according to Property **P3**, the exceeding language generated by the model is the empty set. It can be seen in Figure 2.4 that the increase in the value of k from 1 to 2 eliminates all cycles that are present in the automaton obtained with $k = 1$ in Example 2.6. Thus, according to **P3**, $L_{Exc} = \emptyset$ in this example, if $k = 2$.

It is important to remark that there is a trade-off to be found between the size and accuracy of the identified model. Indeed, by increasing k, the exceeding language is reduced and the model becomes more accurate, improving its fault detection capability. On the other hand, in this case, the number of states is higher, resulting in a more complex model. Some guidelines for the choice of the parameter k are provided in KLEIN *et al.* [19].

In the worst-case scenario in terms of model growth, the number of states in the DAOCT is equal to the sum of the lengths of all observed paths p_{u_i}, $i = 1, \ldots, r$, used in the identification procedure. In this case, $|X| = \sum_{i=1}^{r} l'_i$, which can be obtained by making $k \geq \max_{i \in \{1,\ldots,r\}} l'_i$.

Chapter 3

Timed Automaton with Outputs and Conditional Transitions

In this chapter, the identified timed model is formally introduced and the procedure which is carried out in order to compute the model is presented.

3.1 Observation of the fault-free system behavior

The identification procedure for obtaining the proposed timed identified model is based on the procedure used in Section 2.5 to obtain the DAOCT model. The main difference is that, now, the time delays associated with each event occurrence must also be recorded during identification.

In Section 2.5, u_j denotes the I/O vector reached by the system at a time instant t_j, $j \in \mathbb{N}$. Then, the I/O vector remains unchanged until the occurrence of event σ_j, leading to a new I/O vector u_{j+1}. Now, let us also define the sojourn time τ_j as the time during which the I/O vector remained equal to u_j, *i.e.*, $\tau_j = t_{j+1} - t_j$. Then, when the observed I/O vector changes from u_j to u_{j+1}, we say that *timed transition* $(u_j, \sigma_j, \tau_j, u_{j+1})$ is observed. A finite sequence of such transitions constitutes an observed timed path $p = (u_1, \sigma_1, \tau_1, u_2, \sigma_2, \tau_2, ..., u_{l-1}, \sigma_{l-1}, \tau_{l-1}, u_l)$ executed by the system. If the timing information is removed from $(u_j, \sigma_j, \tau_j, u_{j+1})$, then *untimed transition* (u_j, σ_j, u_{j+1}) is obtained. If all timing information is removed from timed path p, we obtain the untimed path $p_u = (u_1, \sigma_1, u_2, \sigma_2, \ldots, u_{l-1}, \sigma_{l-1}, u_l)$. The length of a timed path is defined to be equal to the length of its associated untimed path.

In order to reduce the number of false alarms, as in Section 2.5, it is assumed that the system has been observed for a sufficiently long time such that all different untimed paths of length up to a given number $n_0 \in \mathbb{N}$ have been observed.

Assume that ν timed paths (not necessarily distinct) have been observed, where each timed path is denoted by

$$p_q = \left(u_{q,1}, \sigma_{q,1}, \tau_{q,1}, u_{q,2}, \sigma_{q,2}, \tau_{q,2}, ..., u_{q,l_q-1}, \sigma_{q,l_q-1}, \tau_{q,l_q-1}, u_{q,l_q}\right), \tag{3.1}$$

where $q \in \{1, ..., \nu\}$, $u_{q,j} \in \Omega$ for $j = 1, ..., l_q$, and $\sigma_{q,j} \in \Sigma$ and $\tau_{j,q} \in \mathbb{R}^+$ for $j = 1, ..., l_q - 1$. Then, the observed timed sequence s_t^q associated with each path p_q is given by $s_t^q = (\sigma_{q,1}, \tau_{q,1})(\sigma_{q,2}, \tau_{q,2})...(\sigma_{q,l_q-1}, \tau_{q,l_q-1})$. Thus, the observed timed language $L_{t,Obs}$ is defined as follows:

$$L_{t,Obs} := \bigcup_{q=1}^{\nu} \overline{\{s_t^q\}}. \tag{3.2}$$

Let P denote the set formed of all observed timed paths p_q, for $q = 1, \ldots, \nu$, and let P be partitioned as $P = P_1 \dot{\cup} P_2 \dot{\cup} \ldots \dot{\cup} P_r$, $r \leq \nu$, such that all paths that form P_i are logically equivalent to each other, *i.e.*, there is a unique untimed path $p_{u_i} = (u_{i,1}, \sigma_{i,1}, u_{i,2}, \sigma_{i,2}, ..., u_{i,l_i'-1}, \sigma_{i,l_i'-1}, u_{i,l_i'})$, associated with each set P_i, for $i = 1, \ldots, r$.

The *time-interval paths* are defined as follows:

$$p_i' = \left(u_{i,1}, \sigma_{i,1}, I_{i,1}, ..., u_{i,l_i'-1}, \sigma_{i,l_i'-1}, I_{i,l_i'-1}, u_{i,l_i'}\right), \tag{3.3}$$

for every $i \in \{1, \ldots, r\}$. The logical behavior of p_i' is given by the untimed path p_{u_i} associated with P_i. The timing behavior associated with path p_i' is modeled by sets $I_{i,j}$, $j = 1, \ldots, l_i' - 1$, which represent the possible time values for the occurrence of events of each one of the transitions of p_{u_i}, obtained from the observed timed paths of P_i. It is assumed that each untimed path, representing a possible logical behavior of the system, has been observed a number of times sufficient to capture the timing information regarding event occurrences. The process of building the sets $I_{i,j}$ from the timed paths in P_i is explained in Section 3.3. It is important to remark that the TAOCT model is computed from the time-interval paths p_i', $i = 1, \ldots, r$, identified from the timed paths $p_q \in P$ generated by the system. The set of time-interval paths is denoted as $P' := \{p_1', p_2', \ldots, p_r'\}$.

3.2 Presentation of the model

The timed model proposed in this work for the identification of DES is presented in the sequel. The model is based on the Timed Automaton with Timing Intervals presented in Definition 2.5, and also on the DAOCT model presented in Definition 2.8.

Definition 3.1 *The Timed Automaton with Outputs and Conditional Transitions* *is a ten-tuple:*

$$TAOCT = (X, \Sigma, f, c_g, \Omega, \lambda, R, g, x_0, X_f),$$

where

- X *is the finite set of states;*
- Σ *is the finite set of events;*
- $f : X \times \Sigma^\star \to X$ *is the transition function;*
- c_g *is the global clock, with value* $c_g(t) \in \mathbb{R}^+$*,* $t \in \mathbb{R}^+$*;*
- $\Omega \subseteq \mathbb{N}_1^{m_i+m_o}$ *is the set of outputs;*
- $\lambda : X \to \Omega$ *is the state output function;*
- $R = \{1, 2, ..., r\}$ *is the set of time-interval path indexes;*
- $g : X \times \Sigma \times R \to \mathcal{C}$ *is the guard function;*
- $x_0 \in X$ *is the initial state;*
- $X_f \subseteq X$ *is the set of final states.* $\square$

The set of admissible constraints $\mathcal{C}$ is formed of all sets $I \subset \mathbb{R}^+$. As in the TATI, presented in Definition 2.5, in the TAOCT a unique global clock c_g is used, and it is reset every time a transition is executed. Function $g(x, \sigma, i)$ specifies a subset of $\mathbb{R}^+$ to which the clock value $c_g(t)$ must belong so that transition (x, σ, x'), where $x' = f(x, \sigma)$, associated with path p'_i, can occur. The output function λ is the same presented in the DAOCT model, and associates an I/O vector with each state $x \in X$ of the model.

Differently from the DAOCT, where the path estimation function θ_u is defined in the model, in the TAOCT the path estimation function $\theta : X \times \Sigma \times \mathbb{R}^+ \to 2^R$ can be defined using the guard function g as follows:

$$\theta(x, \sigma, \tau) = \{i \in R : \tau \in g(x, \sigma, i)\}. \tag{3.4}$$

Function $\theta(x, \sigma, \tau)$ provides the path estimate when the current state is x, and event σ occurs at clock value $c_g(t) = \tau$. If $f(x, \sigma)$ is not defined for a given path p'_i, then $g(x, \sigma, i)$ is, by convention, the empty set, and $i \notin \theta(x, \sigma, \tau)$ for any value of τ. Moreover, if $f(x, \sigma)$ is defined, but the observed time τ does not belong to $g(x, \sigma, i)$, then $i \notin \theta(x, \sigma, \tau)$.

Function $\phi : \Sigma_t^\star \to \Sigma^\star$ removes the timing information from a timed sequence in $\Sigma_t^\star$, obtaining its equivalent untimed sequence, *i.e.*, for any $s_t = (\sigma_1, \tau_1) \dots (\sigma_l, \tau_l) \in \Sigma_t^\star$, then $\phi(s_t) = \sigma_1 \dots \sigma_l \in \Sigma^\star$. By convention, $\phi(\varepsilon) := \varepsilon$.

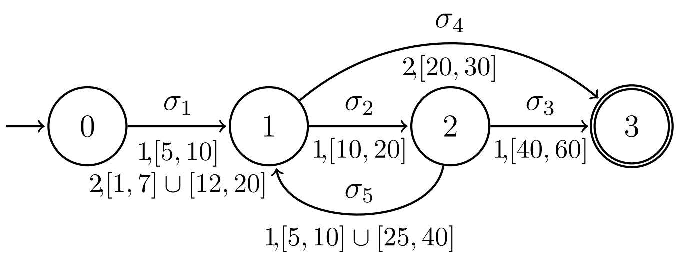


Figure 3.1: TAOCT model of Example 3.1.

The extended path estimation function $\theta_e : X \times \Sigma_t^\star \to 2^R$ is defined, recursively, as follows: $\theta_e(x, \varepsilon) = R$, and for any sequence $s_t(\sigma, \tau) \in \Sigma_t^\star$, where $s_t \in \Sigma_t^\star$ and $(\sigma, \tau) \in \Sigma_t$,

$$\theta_e(x, s_t(\sigma,\tau)) = \begin{cases} \theta_e(x, s_t) \cap \theta(x', \sigma, \tau), \text{ where} \\ \qquad x' = f(x, \phi(s_t)), \text{ if } f(x, \phi(s_t)\sigma)! \\ \emptyset, \text{ otherwise.} \end{cases}$$

The timed language generated by the TAOCT model is given by:

$$L_{t,Iden} := \{s_t \in \Sigma_t^\star : \theta_e(x_0, s_t) \neq \emptyset\}. \tag{3.5}$$

In the sequel, an example of a TAOCT model is presented, and it is discussed how timed event sequences can be generated by the model.

Example 3.1 *Consider the TAOCT shown in Figure 3.1. Each circle represents a state $x \in X$, where $X = \{0, 1, 2, 3\}$, and the directed arcs represent the transitions. A transition from state x to x' is labeled with an event $\sigma \in \Sigma$, where $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$. In addition, a label of the form i, I attached to a transition from x to x' indicates that there is a guard $g(x, \sigma, i) = I$ defined for that transition. In this example, the set of time-interval path indexes is given by $R = \{1, 2\}$, which means that the TAOCT model has been computed from two observed time-interval paths p_i', $i = 1, 2$. The initial state is given by $x_0 = 0$, and the set of final states is $X_f = \{3\}$, represented in Figure 3.1 with a double circle.*

Suppose that one wants to verify if the timed sequence $s_t = (\sigma_1, 6)(\sigma_2, 14)(\sigma_3, 56)$ can be executed by the model. It can be verified that $\theta_e(0, (\sigma_1, 6)) = \{1, 2\}$, since $6 \in g(0, \sigma_1, 1) = [5, 10]$, and $6 \in g(0, \sigma_1, 2) = [1, 7] \cup [12, 20]$. In addition, $\theta_e(0, (\sigma_1, 6)(\sigma_2, 14)) = \theta_e(0, (\sigma_1, 6)) \cap \theta(1, \sigma_2, 14) = \{1\}$, since $14 \in g(1, \sigma_2, 1) = [10, 20]$, and $\theta_e(0, (\sigma_1, 6)(\sigma_2, 14)(\sigma_3, 56)) = \theta_e(0, (\sigma_1, 6)(\sigma_2, 14)) \cap \theta(2, \sigma_3, 56) = \{1\}$, since $56 \in g(2, \sigma_3, 1) = [40, 60]$. Thus, $\theta_e(0, s_t) = \{1\}$ and, according to Equation (3.5), $s_t \in L_{t,Iden}$.

Consider now sequence $s_t' = (\sigma_1, 15)(\sigma_4, 14)$. Then, $\theta_e(0, (\sigma_1, 15)(\sigma_4, 14)) = \theta_e(0, (\sigma_1, 15)) \cap \theta(1, \sigma_4, 14)$. Since, $\theta_e(0, (\sigma_1, 15)) = \{2\}$, and $\theta(1, \sigma_4, 14) = \emptyset$, then

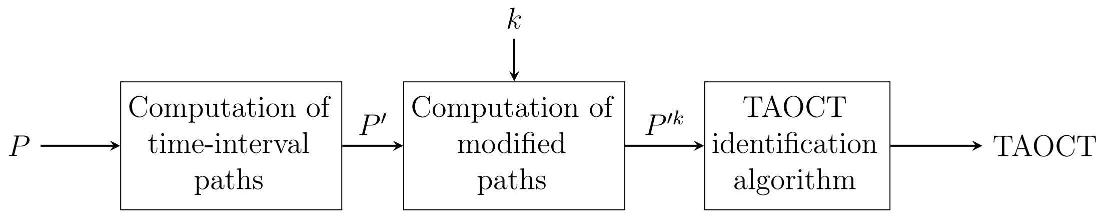

k
P
Computation of time-interval paths
P'
Computation of modified paths
P'^k
TAOCT identification algorithm
TAOCT

Figure 3.2: Scheme of the identification procedure.

$\theta_e(0, s'_t) = \emptyset$. *Thus, according to Equation (3.5),* $s'_t \notin L_{t,Iden}$. □

Sections 3.3 and 3.4 show the steps that are necessary to compute the TAOCT model. In Figure 3.2, the complete scheme of the identification procedure with three steps is depicted. It can be seen from Figure 3.2 that the input data for identification is set $P = \{p_1, p_2, \ldots, p_\nu\}$, containing all observed fault-free timed paths, and the output is the TAOCT that models the system fault-free behavior.

3.3 Computation of the time-interval paths

In order to compute the time-interval paths p'_i, $i = 1, \ldots, r$, it is necessary to obtain the sets $I_{i,j}$ corresponding to the transitions of p'_i, associated with the time values that the transitions of paths $p_q \in P_i$ have been observed.

Let $T_{i,j} := \{\tau_{q,j} \in \mathbb{R}^+ : (q \in \{1, \ldots, \nu\}) \wedge (p_q \in P_i)\}$ be the set formed of the time values $\tau_{q,j}$ that are observed in the j-th transition of the timed paths $p_q \in P_i$. It is possible that the values in $T_{i,j}$ are arranged on the real axis in such a way that some values in a subset of $T_{i,j}$ are closer to each other in comparison with the other values in $T_{i,j}$. Each such subset forms a *cluster* of time values, and each cluster corresponds to a different observed timing behavior for that transition. Thus, different timing behaviors can be identified by applying a clustering method to the values in $T_{i,j}$.

Several clustering algorithms have been proposed in the literature [36–39]. In the sequel, two different methods are proposed for obtaining the clusters of time values in each $T_{i,j}$. The first one is based on a clustering method called Nearest Neighbors Method, and the second one is inspired by a classical tool in the field of unsupervised learning: the K-Means clustering method.

3.3.1 The Nearest Neighbors Method

This method is based on the Nearest Neighbors Method [37] for obtaining the clusters of $T_{i,j}$. In this method, when adapted to the one-dimensional case, two successive values belong to the same cluster if the distance between them is less than or equal to a user specified threshold ζ. If the distance between them is greater than ζ, then they are assigned to different clusters.

Note that an uncertainty $\delta \in \mathbb{R}^+$ must be considered for defining the value of ζ to take into account possible errors in measuring the time values of the event occurrences, caused, for example, by the scan cycle of the controller. Due to this uncertainty, the threshold ζ must be chosen so that $\zeta > 2\delta$.

After applying the Nearest Neighbors Method to $T_{i,j}$, set $S = \{S_1 \ldots, S_K\}$ is obtained, such that $S_1 \dot{\cup} S_2 \dot{\cup} \ldots \dot{\cup} S_K = T_{i,j}$. Each set $S_h \in S$ corresponds to a different cluster in $T_{i,j}$, representing a distinct timing behavior associated with the j-th transition of time-interval path p'_i. Set $I_{i,j}$ is then given by the following expression:

$$I_{i,j} = \bigcup_{h=1}^{K} [\max\{0, \min S_h - \delta\}, \max S_h + \delta].$$

Note that set $I_{i,j}$ is a disjoint union of intervals, where each interval corresponds to a cluster S_h. Each interval is extended by δ on both ends in order to reduce the number of false alarms that may be raised due to disturbances in time measurements. Since only non-negative values are allowed in $I_{i,j}$, if the minimum value of a cluster is less than δ (*i.e.*, $\min S_h - \delta < 0$), then the lower endpoint of the interval corresponding to this cluster is set to zero.

For the choice of the threshold ζ, the following considerations must be taken into account. Let ζ be chosen to be a large value such that two distinct timing behaviors of *time-interval transition* $(u_{i,j}, \sigma_{i,j}, I_{i,j}, u_{i,j+1})$ of path p'_i are merged into a single cluster, *i.e.*, $I_{i,j}$ is formed of a single time interval. In this case, if a fault in the system causes the transition to occur between the two distinct timing behaviors, then the fault is not detected, since this time value belongs to $I_{i,j}$, which means that it is included in the fault-free model. On the other hand, if time-interval transition $(u_{i,j}, \sigma_{i,j}, I_{i,j}, u_{i,j+1})$ of path p'_i has a unique timing behavior, but ζ is chosen to be a small value and the number of observations of the transition is small, then it is possible that several clusters are formed instead of a single cluster. As a consequence, $I_{i,j}$ is formed of several time intervals, and a false alarm is raised every time transition $(u_{i,j}, \sigma_{i,j}, u_{i,j+1})$ is observed between the time intervals of $I_{i,j}$. Thus, for the correct computation of the clusters associated with $I_{i,j}$, ζ must be a small value greater than 2δ, and the system must be observed for a sufficiently long time such that there are enough observations of each transition of p'_i to adequately represent its distinct timing behaviors. It is worth noting that in the ideal case where infinite observations of the same transition are recorded, the distance that separates two successive values in the same cluster would drop to zero. Thus, it is reasonable to choose a small value for ζ, provided that a large amount of data recorded during identification is available. Nevertheless, if false alarms are raised during system operation, the model can still be improved by increasing the value of

ζ for the computation of each $I_{i,j}$.

It is important to remark that the clustering procedure must be performed for the determination of each set $I_{i,j}$ from set $T_{i,j}$, for every time-interval path p'_i, $i \in \{1, \dots, r\}$.

In the sequel, an example of the formation of a time-interval path from observed timed paths is presented.

Example 3.2 *Suppose that paths of the following form have been observed 200 times during the identification process:*

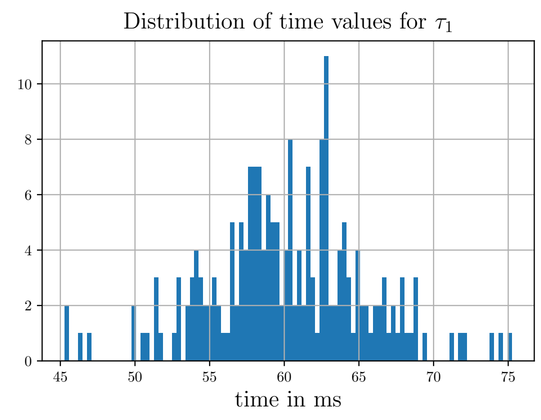
$$p = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a, \tau_1, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, \tau_2, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, c, \tau_3, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right).$$

Assume that: (i) the time values τ_1 for each observed path are normally distributed, and that the mean and standard deviation of this distribution are equal to 60ms and 5ms, respectively; (ii) the time values τ_2 follow a mixture distribution of three normal curves, where the means of each curve are given by 20ms, 200ms and 600ms, and the standard deviations are given by 2.5ms, 10ms and 30ms, respectively; and (iii) the values of τ_3 were sampled from a mixture distribution of two normal curves, with means given by 310ms and 1000ms, and the standard deviation is equal to 50ms for both curves. In Figure 3.3, the distributions of values for τ_1, τ_2, and τ_3 are shown. In this example, the time-interval path p'_1 is obtained from the observed paths. By selecting $\zeta = 50ms$ (since this value seems like a good choice given the parameters of the time distributions) in the Nearest Neighbors Method and considering $\delta = 20ms$, the following single time-interval path is obtained:

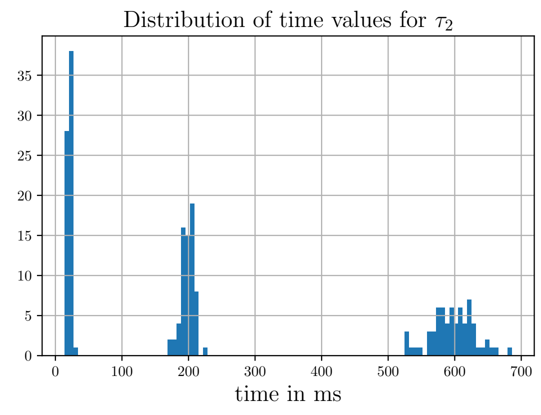
$$p'_1 = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a, I_{1,1}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, I_{1,2}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, c, I_{1,3}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right),$$

where $I_{1,1} = [25.3, 95.2]$, $I_{1,2} = [0, 47.7] \cup [148.8, 246.4] \cup [505.2, 705.9]$ and $I_{1,3} = [165.9, 481.3] \cup [818.9, 1133.6]$. The minimum and maximum time values in Figure 3.3a are 45.3ms and 115.2ms, respectively. Since the difference between two consecutive values in the distribution for τ_1 is always lower than 50ms, then set $I_{1,1}$ is determined as the single interval $[45.3 - \delta, 115.2 + \delta]$, given that $45.3 - \delta > 0$. Sets $I_{1,2}$ and $I_{1,3}$ are obtained in a similar fashion, with the difference being that, in these sets, there are consecutive values which are more than 50ms apart, leading to the formation of clusters. □

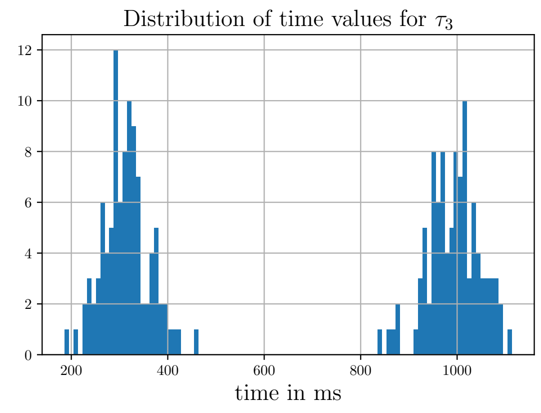
In the sequel, an alternative approach for obtaining sets $I_{i,j}$ is proposed. The main difference of this method with respect to the Nearest Neighbors Method is



(a) τ_1



(b) τ_2



(c) τ_3

Figure 3.3: Histograms showing the distributions of time values presented in Example 3.3.

that a user-specified threshold for cluster separation does not need to be provided, which means that it is not necessary to have any previous knowledge about the time dynamics of the system under study.

3.3.2 Clustering method based on K-Means

A classical method used for clustering is the K-Means method [36, 38]. The K-Means algorithm can be summarized by the following steps [38].

Step 1. Choose K cluster centers to coincide with K randomly defined points inside the region containing all the points;

Step 2. Assign each point to the cluster defined by the closest cluster center;

Step 3. Recompute the cluster centers using the points that are currently assigned to each cluster;

Step 4. If there has been no (or minimal) reassignment of points to other clusters, then stop algorithm. Otherwise, go to Step 2.

In order to apply the K-Means algorithm, it is necessary to provide the desired number of clusters K, which can be a problem when the number of clusters which best fits the time values for each observed transition is not known in advance. Hence, estimation of K is required prior to applying the K-Means method to the time values associated with a given observed transition. Some methods are proposed in the literature for estimating the number of clusters K [39, 40]. In the sequel, an adaptation of the method presented in PHAM *et al.* [39] is proposed for the case of one-dimensional data, *i.e.*, where the values to be clustered lie on the real axis.

After applying the K-Means algorithm to $T_{i,j}$, K clusters $S_h \subseteq T_{i,j}$, $h = 1, \ldots, K$, are obtained. Let τ_v^h, $v \in \{1, 2, \ldots, |S_h|\}$, denote a time value in cluster S_h, and τ_{avg}^h denote the average value of all $\tau_v^h \in S_h$. Then, the total distortion is defined as:

$$d_K := \sum_{h=1}^{K} \sum_{v=1}^{|S_h|} (\tau_v^h - \tau_{avg}^h)^2,$$

Note that the total distortion decreases monotonically with increasing values of K, since the rise in the number of cluster centers reduces the distances between a center and the points in the same cluster, which decreases the contribution of each cluster to the value of d_K.

Let parameter $K_{max} \in \mathbb{N} \backslash \{0\}$ be the user-specified predefined maximum number of clusters, the choice of which will be explained later on. Based on a similar definition presented in PHAM *et al.* [39], we introduce a measure given by the evaluating function $\mathcal{E} : \{1, 2, \ldots, K_{max}\} \to [0, 1]$, calculated as follows:

$$\mathcal{E}(K) := \begin{cases} 1, & \text{if } K = 1 \\ \frac{d_K}{d_{K-1}}, & \text{if } 1 < K \leq K_{max} \end{cases}.$$

Function $\mathcal{E}(K)$ measures the relative decrease in the total distortion d_K by increasing the number of clusters from $K-1$ to K. A significant decrease in the value of d_K with respect to d_{K-1}, which results in a low value for $\mathcal{E}(K)$, suggests that K is a good estimation for the number of clusters in the data [39]. Here, a similar strategy is adopted in order to determine the number of clusters K.

The proposed clustering procedure can be described as follows. There are three parameters that must be defined:

- the maximum number of desired clusters $K_{max} \geq 1$, which is assumed to be much lower than the total number of values to be clustered (for any transition);
- the free parameter $\xi \in [0,1]$;
- the uncertainty $\delta \in \mathbb{R}^+$.

At first, if $K_{max} > 1$, we determine the value $K' \in \{2,3,\dots,K_{max}\}$ for which the total distortion is minimized and call this value K^*. If $\mathcal{E}(K^*)$ is greater than the free parameter ξ, it means that the points are already relatively grouped together and the decrease in the total distortion is not expressive enough to justify the formation of clusters. In this scenario, as well as in the case where $K_{max} = 1$, let us define a variable $\tilde{S}$ and make $\tilde{S} = \{T_{i,j}\}$. On the other hand, if $\mathcal{E}(K^*) \leq \xi$ and $K_{max} > 1$, then the K-Means method is applied to the input set $T_{i,j}$ specifying K^* as the desired number of clusters, resulting in set $S' = \{S'_1, \dots, S'_{K^*}\} \subseteq 2^{T_{i,j}}$, with each $S'_h \in S'$ being a different cluster, and $S'_{h'} \cap S'_{h''} = \emptyset$, for all $S'_{h'}, S'_{h''} \in S'$ such that $h' \neq h''$. The uncertainty δ can be considered in order to take into account possible errors in measuring the time occurrence of the events, as discussed in the case of the Nearest Neighbors Method. In this case, if the distance between any two clusters falls under 2δ, then these two clusters that were previously obtained are merged, given that the uncertainty introduced by the scan cycle does not justify the separation of clusters. The resulting set after performing the merge operations is then $\tilde{S} = \{\tilde{S}_1, \dots, \tilde{S}_K\}$, with $1 \leq K \leq K^*$. This expression for $\tilde{S}$ also holds in the case where $\mathcal{E}(K^*) > \xi$, by making $K = 1$ and $\tilde{S}_1 = T_{i,j}$. Set $I_{i,j}$ is then given by the following expression:

$$I_{i,j} = \bigcup_{h=1}^{K} [\max\{0, \min \tilde{S}_h - \delta\}, \max \tilde{S}_h + \delta]. \tag{3.6}$$

The value of the number of clusters K must be bounded by $K_{max} << \min\{|T_{i,j}| : i \in \{1,\dots,r\}$ and $j = 1,\dots,l'_i - 1\}$ so that the algorithm does not search for high numbers of clusters, since it could give clusters that are too specialized, with few values assigned to each cluster. Moreover, for higher values of K_{max}, the computational cost of the cluster function can increase substantially, given that it is necessary to apply the K-Means algorithm for every $K \in \{1, \ldots, K_{max}\}$ in the process of finding the best estimate for the number of clusters. This is a problem that must be considered, especially in the case where computational resources are limited.

The role of the parameter ξ is to allow the formation of clusters only if the decrease in the total distortion is significant. If ξ is too low, then cluster formation does not occur and all time values associated with a given transition are assigned to the same cluster. As a result, for that transition, any observed timing that falls between the endpoints of that cluster is considered to be part of the fault-free behavior. If, in the real system, more than one cluster are required to adequately model the timing behavior for that transition, then more fault occurrences will be missed by the detector, since a fault may cause the timing of the transition to fall between two clusters that were supposed to be formed. If, on the other hand, the chosen value of ξ is too high, then the tendency is that clusters are formed when, in reality, the time values are already relatively well packed together. In this case, even if no fault has occurred, there is a risk that the timing of a transition may fall between two clusters that should not have been formed. Thus, the detector would raise a false alarm, as a consequence of the clusters being overfitted. Therefore, the choice of the parameters K_{max} and ξ must be made considering the trade-off between missed detection of faults and the risk of false alarms raised by the fault detector.

In the sequel, we present an example of the formation of a time-interval path from observed timed paths using the clustering approach based on the K-Means Method.

Example 3.3 *Let us consider the same observed paths that were presented in Example 3.2. In this example, we wish to obtain the time-interval paths from the observed paths using the clustering method based on the K-Means procedure. As in Example 3.2, since the logical sequence of events are shared by all observed paths, there is only one time-interval path with the same event sequence. By selecting $K_{max} = 5$, $\xi = 0.2$ and $\delta = 20ms$, the single time-interval path is given by:*

$$p'_1 = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a, I_{1,1}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, I_{1,2}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, c, I_{1,3}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right),$$

where $I_{1,1} = [25.3, 95.2]$, $I_{1,2} = [0, 47.7] \cup [148.8, 246.4] \cup [505.2, 705.9]$ and $I_{1,3} = [165.9, 481.3] \cup [818.9, 1133.6]$. These sets were obtained by analyzing the evolution of function $\mathcal{E}(K)$ from $K = 1$ up to $K = K_{max} = 5$, which can be seen in Figure 3.4. Note that this result is identical to the one obtained with the Nearest Neighbors Method (see Example 3.2), since the number of clusters in each transition is lower *than K_{max}. The following remarks can be made:*

- *In Figure 3.4a, we can see that, even though $\mathcal{E}(K)$ reaches its minimum at $K = 2$, its value is approximately 0.4, which is higher than the chosen value for ξ, which is 0.2. For this reason, only one cluster has been found to fit the distribution of time values shown in Figure 3.3a.*
- *In Figure 3.4b, the minimum of $\mathcal{E}(K)$ is found at $K = 3$. It is interesting to notice, however, that $\mathcal{E}(2)$ is close to the minimum. This reflects the fact that the distribution of time values can almost be seen as being composed of two clusters, which is justified by the closer proximity of the first two clusters (from left to right in Figure 3.3b) in comparison with the third one. This mirrors the result provided by the method, given that, since $\mathcal{E}(2)$ is slightly higher than $\mathcal{E}(3)$, then $K = 2$ is almost as good a choice as $K = 3$. In fact, three clusters can be easily identified by a visual inspection of Figure 3.3b, which means that the method indeed seems to provide the best estimate for the number of clusters.*
- *Since the decrease in the value of $\mathcal{E}(K)$ from $K = 1$ to $K = 2$ is sharp enough to fall below the value of ξ (see Figure 3.4c), two clusters are identified in the distribution shown in Figure 3.3c. This matches the behavior of the points in Figure 3.3c, where two distinct clusters of data can be easily spotted.* $\square$

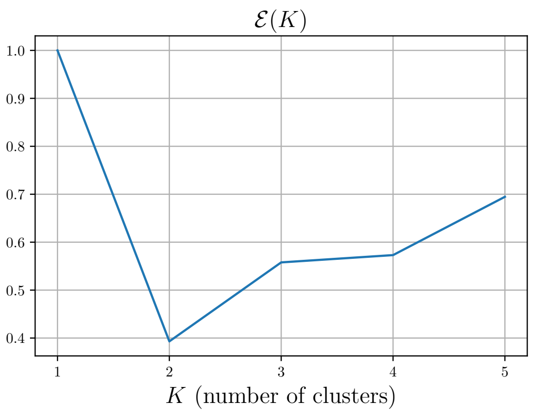
3.4 Computation of the TAOCT model

It is assumed in this section that the sets $I_{i,j}$ of the time-interval paths p'_i, $i \in \{1, \dots, r\}$ have been determined by using one of the methods that are presented in Section 3.3.

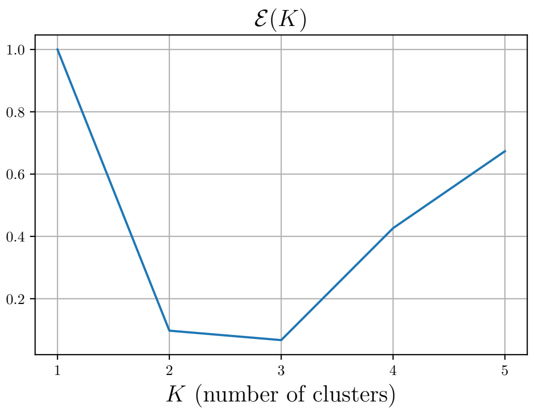
In Section 2.5, the introduction of the free parameter k allows to obtain a model satisfying Properties **P2** and **P3**. Since the TAOCT is based on the DAOCT model presented in Section 2.5, the same strategy can be used to obtain a parameterized model for identification. As a result, the TAOCT inherits Properties **P2** and **P3** from its underlying DAOCT. Therefore, as far as its logical behavior is concerned, the obtained TAOCT model is k-complete (see Definition 2.7) and the absence of cycles for a given k implies that the exceeding language generated by the model is the empty set. The choice of k must be made by considering the trade-off between complexity and accuracy of the identified model, as explained in Section 2.5.

The same steps for the computation of the DAOCT model are considered here, where the first step is the computation of timed modified paths p'^k_i, according to the free parameter k, as follows:

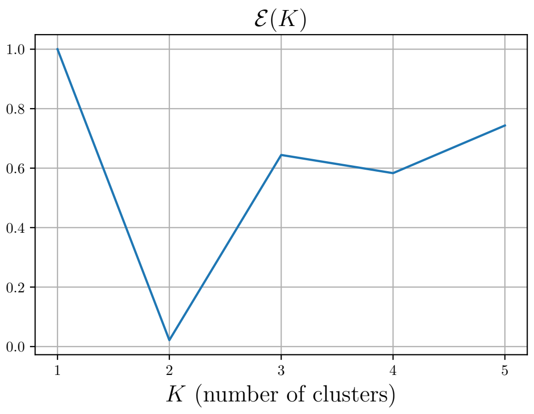
$$p'^k_i = \left(y_{i,1}, \sigma_{i,1}, I_{i,1}, ..., y_{i,l'_i-1}, \sigma_{i,l'_i-1}, I_{i,l'_i-1}, y_{i,l'_i}\right), \tag{3.7}$$



(a) τ_1



(b) τ_2



(c) τ_3

Figure 3.4: Values of $\mathcal{E}(K)$, for $K = 1, \dots, K_{max}$, for each set of time values corresponding to τ_1, τ_2 and τ_3, presented in Example 3.3.

where

$$y_{i,j} = \begin{cases} (u_{i,j-k+1}, ..., u_{i,j}), & \text{if } k \leq j \leq l'_i \\ (u_{i,1}, ..., u_{i,j}), & \text{if } j < k \end{cases}.$$

Define set Ω^k as the set formed of all $y_{i,j}$, $j = 1, \dots, l'_i$, $i = 1, \dots, r$. Let $\hat{y}_{i,j}$ denote the last element of $y_{i,j}$. Then, $\hat{y}_{i,j} := u_{i,j}$, for $j = 1, \dots, l'_i$, $i = 1, \dots, r$. The set of timed modified paths is denoted as $P'^k := \{p'^k_1, p'^k_2, \dots, p'^k_r\}$.

Before introducing the procedure to construct the TAOCT from the observed data, define function $\tilde{\lambda} : X \to \Omega^k$, which provides a bijective correspondence between a state $x \in X$ of the model and a symbol $y \in \Omega^k$. This function will be used in the procedure for constructing the TAOCT model from the set of fault-free modified paths $p_i'^k$, described in Algorithm 3.1.

Since the TAOCT model is based on the DAOCT, Algorithm 3.1 is based on the algorithm for computing the DAOCT model presented in MOREIRA and LESAGE [21]. At the beginning of Algorithm 3.1, the initial state x_0 is created, and $\tilde{\lambda}(x_0)$ and $\lambda(x_0)$ receive $y_{1,1}$ and $\hat{y}_{1,1}$, respectively, such that the output of the initial state is equal to the first I/O vector of the observed paths $p_q \in P$, used for the computation of the modified time-interval paths $p_i'^k$, for $i = 1, \ldots, r$. In line 3, the set of states X, the set of outputs Ω, and the set of final states X_f are initialized. In addition, the set of time-interval path indexes R is specified. In line 4, the set of events Σ is defined as the set containing all observed events, considering every transition in each observed fault-free time-interval path p_i', for $i = 1, \ldots, r$. In line 5, guard function g is initialized for state x_0 by assigning the empty set for every observed event in Σ and every path index in R. In the inner for-loop, from lines 7 to 21, every vertex $y_{i,j}$ of the modified time-interval path $p_i'^k$ is visited. Each time a vertex is visited, the corresponding state $x \in X$ is found. Then, it is checked if there is a state x' that has already been created and that corresponds to the next vertex $y_{i,j+1}$. If such a state does not exist, then a new state is created in the if-block that starts in line 9, and set X is updated accordingly. In lines 12 to 14, vertex $y_{i,j+1}$ is associated to the newly created state x', and its output symbol, given by the last element of $y_{i,j+1}$, is added to Ω. Similarly to what is done in line 5 for the initial state x_0, in line 15 the empty set is assigned to the guard function for state x' and every pair (σ, i) in $\Sigma \times R$, ensuring that the guard function is completely defined on its domain. In line 18, the transition from the current state x to the next state x' through event $\sigma_{i,j}$ is defined in the transition function f. Differently from g, function f may be partially defined on its domain. In line 19, the guard function $g(x, \sigma_{i,j}, i)$ is updated by aggregating set $I_{i,j}$ to $g(x, \sigma_{i,j}, i)$. Finally, in line 20, it is tested if the next vertex $y_{i,j+1}$ is the last one of path $p_i'^k$. If this is the case, then next state x' is added to the set of final states X_f.

Language $L_{t,Iden}$, generated by the TAOCT model obtained using Algorithm 3.1, and whose expression is given by Equation (3.5), simulates the timed sequences of the paths used in the identification procedure, as stated in the sequel.

Theorem 3.1 *Suppose that the language $L_{t,Iden}$ generated by the TAOCT model, whose expression is given by Equation* (3.5)*, is obtained using Algorithm 3.1. Then:*

$$L_{t,Obs} \subseteq L_{t,Iden}.$$

Algorithm 3.1: TAOCT identification

```
Input: Modified time-interval paths $p_i'^k$, $i = 1, ..., r$
Output: TAOCT $= (X, \Sigma, f, c_g, \Omega, \lambda, R, g, x_0, X_f)$
1  Create initial state $x_0$
2  $\lambda(x_0) \leftarrow \hat{y}_{1,1}$ and $\tilde{\lambda}(x_0) \leftarrow y_{1,1}$
3  $X \leftarrow \{x_0\}$, $\Omega \leftarrow \{\hat{y}_{1,1}\}$, $R \leftarrow \{1, ..., r\}$, and $X_f \leftarrow \emptyset$
4  $\Sigma \leftarrow \bigcup_{i \in R} \{\sigma_{i,j} : j = 1, \ldots, l_i' - 1\}$
5  $g(x_0, \sigma, i) \leftarrow \emptyset$, $\forall (\sigma, i) \in \Sigma \times R$
6  for $i = 1$ to $r$ do
7      for $j = 1$ to $l_i' - 1$ do
8          Find state $x \in X$ such that $\tilde{\lambda}(x) = y_{i,j}$
9          if $\tilde{\lambda}(x') \neq y_{i,j+1}$ $\forall x' \in X$ then
10             Create state $x'$
11             $X \leftarrow X \cup \{x'\}$
12             $\tilde{\lambda}(x') \leftarrow y_{i,j+1}$
13             $\lambda(x') \leftarrow \hat{y}_{i,j+1}$
14             $\Omega \leftarrow \Omega \cup \{\lambda(x')\}$
15             $g(x', \sigma, i') \leftarrow \emptyset$, $\forall (\sigma, i') \in \Sigma \times R$
16         else
17             Find $x' \in X$ such that $\tilde{\lambda}(x') = y_{i,j+1}$
18         $f(x, \sigma_{i,j}) \leftarrow x'$
19         $g(x, \sigma_{i,j}, i) \leftarrow g(x, \sigma_{i,j}, i) \cup I_{i,j}$
20         if $j = l_i' - 1$ then
21             $X_f \leftarrow X_f \cup \{x'\}$
```

Proof. Consider a timed event sequence $w_t \in L_{t,Obs}$, which, according to Equation (3.2), is the prefix of a timed sequence associated with some observed timed path p_q, whose expression is given by Equation (3.1). Since $p_q \in P_i$, for some $i \in \{1, \ldots, r\}$, then its corresponding modified time-interval path $p_i'^k$, given by Equation (3.7), has the same untimed sequence as p_q, and p_q is such that $\tau_{q,j} \in I_{i,j}$, for $j = 1, ..., \|w_t\|$. Since $\tilde{\lambda}(x_0) = \hat{y}_{i,1}$ (as a consequence of Assumption **A1**), Algorithm 3.1 ensures that: (i) there exist states x_j, such that $f(x_j, \sigma_{i,j}) = x_{j+1}$, $j = 1, ..., \|w_t\|$, starting at the initial state x_0 (line 18); and (ii) $\tau_{q,j} \in I_{i,j} \subseteq g(x_j, \sigma_{i,j}, i)$, $j = 1, ..., \|w_t\|$, according to line 19. Hence, according to Equation (3.4), $i \in \theta(x_j, \sigma_{i,j}, \tau_{q,j})$, $j = 1, ..., \|w_t\|$, which implies that $i \in \theta_e(x_0, w_t)$, and, consequently, $\theta_e(x_0, w_t) \neq \emptyset$. Thus, from Equation (3.5), $w_t \in L_{t,Iden}$. Therefore, $L_{t,Obs} \subseteq L_{t,Iden}$. □

Let L_{Obs} and L_{un} denote the languages formed of the untimed sequences of events obtained from all timed sequences of $L_{t,Obs}$ and $L_{t,Iden}$, respectively, *i.e.*, $L_{Obs} := \{\phi(s_t) \in \Sigma^\star : s_t \in L_{t,Obs}\}$ and $L_{un} := \{\phi(s_t) \in \Sigma^\star : s_t \in L_{t,Iden}\}$. Then, the following result can be stated.

Corollary 3.1 $L_{Obs} \subseteq L_{un}$.

Proof. The proof is straightforward from Theorem 3.1. $\square$

Chapter 4

Fault detection scheme

The fault detector proposed in this chapter is inspired by the one presented in MOREIRA and LESAGE [22] based on the DAOCT model. Thus, as long as a sequence of timed events s_t, whose corresponding untimed behavior $s_u = \phi(s_t)$ belongs to L_{Obs}, is executed by the system, the fault detector observes the events, and plays the model following the behavior of s_t. If sequence s_t is compatible with one of the time-interval paths p'_i, then, after sequence s_t has been observed, the model is reinitialized and a new sequence can be played by the fault detector. The sequence of events that is played by the fault detector without reinitializing the model is called a *model run*. Thus, the fault detector must evaluate if the current model run can be executed by the TAOCT model. If the model is unable to execute the observed sequence of events, or if the occurrence of an expected event does not occur within a feasible time interval, then a fault is detected.

In Figure 4.1, the scheme for fault detection based on the TAOCT model is presented. The idea behind this scheme is that, during system operation, the current I/O vector of controller signals is read in real-time by the fault detector. Then, if the observed behavior of the system is different from what is predicted by the TAOCT model, the fault that caused the unexpected behavior is detected.

In MOREIRA and LESAGE [22], the minimum number of event observations,

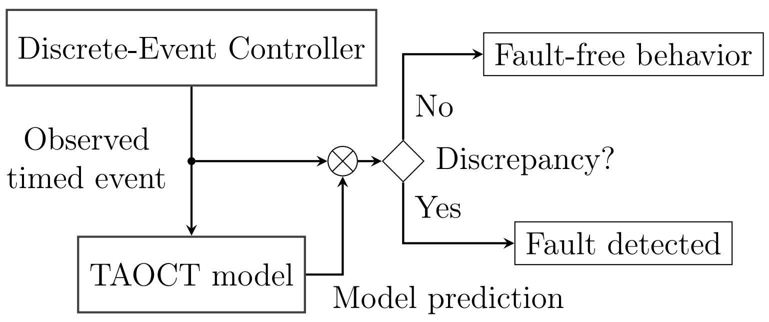


Figure 4.1: Fault detection scheme based on the TAOCT model.

denoted here as μ_{u_i}, such that each fault-free path can be distinguished from the other fault-free paths, is used in the fault detection scheme. Similarly, here the minimum number of event observations of p'_i such that p'_i can be distinguished from any other path p'_z, $i \neq z$, $i, z \in R$, denoted as μ_i, is used in the fault detection scheme. However, due to the fact that the timing behavior of the TAOCT is also considered, the value of μ_i may be different from the value of μ_{u_i}, obtained for the underlying DAOCT model, for each $i \in R$. Consider, for example, that $R = \{1, 2\}$, and that both paths p'_1 and p'_2 start with the same event, *i.e.*, $\sigma_{1,1} = \sigma_{2,1}$. Since p'_1 and p'_2 have the same first event, then they are logically equivalent considering only the first event occurrence, which means that certainly both μ_{u_1} and μ_{u_2} are greater than one by considering only the logical behavior. However, if $I_{1,1} \backslash I_{2,1} \neq \emptyset$, then it is possible to distinguish path p'_1 from p'_2 upon the occurrence of the first event. In order to do so, the first event must be observed with a time value belonging to $I_{1,1} \backslash I_{2,1}$. In this case, $\mu_1 = 1$. Note that, if $I_{2,1} \subset I_{1,1}$, then it is impossible to distinguish p'_2 from p'_1 upon the first event occurrence. Thus, $\mu_2 > 1$. The value of μ_i, for $i = 1, \ldots, r$, can be formally defined as follows:

$$\begin{aligned} \mu_i = \min\{ & j \in \{1, \ldots, l'_i - 1\} : (\nexists z \in R \setminus \{i\}) \\ & [(\sigma_{i,v} = \sigma_{z,v}) \wedge (I_{i,v} \backslash I_{z,v} = \emptyset), v = 1, \ldots, j]\}. \end{aligned} \tag{4.1}$$

It is important to remark that it is always possible to compute μ_i, $i = 1, \ldots, r$, since, according to Assumption **A2**, none of the untimed paths p_{u_i}, associated with p'_i, has an associated sequence of events that is a prefix of the sequence of events of another untimed path p_{u_z}, associated with p'_z, where $i \neq z$, for $i, z \in R$.

In the definition of a viable event given in MOREIRA and LESAGE [22], four conditions are presented in order to verify if an observed event indicates the occurrence of a fault. In the sequel, a *viable timed event* is defined by adapting the definition of viable event to the context of the TAOCT model.

Definition 4.1 *Let $s_t \in \Sigma_t^\star$ be a model run such that $x = f(x_0, \phi(s_t))$. Then, $(\sigma, \tau) \in \Sigma_t$ is said to be a viable timed event in state $x \in X$ of the TAOCT model if it satisfies the following conditions:*

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

C2. *$\theta_e(x_0, s_t(\sigma, \tau)) \neq \emptyset$;*

C3. *If $|\theta_e(x_0, s_t)| > 1$ and $\theta_e(x_0, s_t(\sigma, \tau)) = \{i\}$, then $\|s_t(\sigma, \tau)\| \geq \mu_i$;*

C4. *If $\|s_t(\sigma, \tau)\| = l'_i - 1$, for $i \in \theta_e(x_0, s_t(\sigma, \tau))$, then $\tilde{\lambda}(x') = y_{i,l'_i}$, where $x' = f(x, \sigma)$, or there exists $j \in \theta_e(x_0, s_t(\sigma, \tau))$ such that $\|s_t(\sigma, \tau)\| < l'_j - 1$.* □

If Conditions **C1** and **C2** are verified, then $s_t(\sigma, \tau) \in L_{t,Iden}$. If Condition **C3** is violated, then path p'_i has been wrongly identified before the minimum number of timed event occurrences μ_i, given by Equation (4.1). Finally, if Condition **C4** is not true, then the length of the observed trace $s_t(\sigma, \tau)$ is equal to the maximum length among all sequences of the estimated paths in $\theta_e(x_0, s_t(\sigma, \tau))$, without reaching the final vertex of any of these paths, which implies that a fault has occurred.

Algorithm 4.1 describes the procedure for performing fault detection using the identified TAOCT model. The basic idea is to detect the occurrence of a fault if the observed timed event is not viable, according to Definition 4.1, or a deadlock is reached. Let us define sets $V := \{(i, y_{i,l'_i}) : i \in R\}$ and $N := \{(i, \mu_i) : i \in R\}$, which are used in the algorithm. Verification of Conditions **C1**-**C4** is carried out for the TAOCT in a similar way as for the underlying DAOCT presented in MOREIRA and LESAGE [22]. The main difference between Algorithm 4.1 and the algorithm proposed in MOREIRA and LESAGE [22] is the use of the path estimation function θ_e that takes into account the information about the time that an event is observed, τ, for determining which time-interval paths are possibly being executed by the system. Since θ_e uses both the logical and the timing behaviors of the system, then its estimation is improved with respect to the estimation provided by the path estimation function presented in Section 2.5, which uses only the logical behavior. Thus, the number of faults that can be detected by using the TAOCT is always greater than or equal to the number of faults that can be detected by using the underlying DAOCT.

There are three faulty scenarios for which a fault can be detected by using the TAOCT instead of the underlying DAOCT model, namely:

1. faults that lead the system to a deadlock;

2. faults that cause the occurrence of a feasible event σ at a time instant τ that does not belong to any set defined by the guard conditions $g(x, \sigma, i)$, for all $i \in R$;

3. faults that cause the occurrence of a feasible event σ at a time instant τ that satisfies a guard condition, but leads the path estimation function to $\theta_e(x_0, s_t) = \emptyset$.

In order to detect the first type of fault, the global clock c_g is reset in line 5 every time a transition is transposed in the TAOCT. Then, c_g is used to detect a deadlock in lines 7 to 11, by verifying if an event is observed in a time smaller than or equal to the maximum time τ_{max} allowed for the occurrence of an event in the current state x_c. If an event is not observed within τ_{max} time units, then the fault is detected. The second and third types of faults can be detected by verifying if the observed timed sequence can be generated by the model, since, in both cases, even though $f(x_0, \phi(s_t))$ is defined, $\theta_e(x_0, s_t) = \emptyset$, *i.e.*, the observed timed sequence is not in $L_{t,Iden}$.

In line 13, it is checked if Condition **C1** is violated, in which case the fault is detected in line 14. In lines 15 to 17, Condition **C2** is verified and, if it does not hold, the fault is detected. In lines 18 to 21, Condition **C3** is tested and the fault is detected if this condition is violated. In lines 22 to 25 of Algorithm 4.1, the TAOCT is reinitialized every time a cyclical fault-free path, with behavior compatible with a time-interval path p'_i, $i \in \{1, \ldots, r\}$, is executed by the system. In lines 26 to 30, it is verified if the final vertex of a non-cyclical observed path is reached, in which case no event should be observed anymore. In this case, in line 28 the path estimate is made equal to the empty set, in line 29 the value of τ_{max} is set to infinity, and the algorithm returns to line 5. Then, if an event is observed, the fault is detected in lines 13 and 14, since Condition **C1** is violated. In line 31, Condition **C4** is tested and, if it is violated, the fault is detected in line 32. If the end of the path that is being executed by the system has not been reached yet, then the path estimation function and the state of the model are updated in lines 33 to 37, and the algorithm returns to line 4.

The concept of reinitializability is introduced in MOREIRA and LESAGE [22] to present a condition for ensuring that the model can always be reinitialized after the completion of some fault-free path. This condition is restated for the TAOCT model as follows.

Definition 4.2 *Let s_t denote an observed sequence of timed events compatible with path p'_i, i.e., $i \in \theta_e(x_0, s_t)$ and $\|s_t\| = l'_i - 1$, for some $i \in R$. Then, the TAOCT is said to be reinitializable if there does not exist $s'_t \in \overline{\{s_t\}}$ of length $\|s'_t\| = l'_z - 1$, where $z \in \theta_e(x_0, s'_t)$ and $l'_z < l'_i$, such that $x' = f(x_0, \phi(s'_t))$, and $\tilde{\lambda}(x') = y_{z,l'_z}$.* □

In order to clarify the meaning of the concept of reinitializability, let us consider that a path p'_i is executed by the system and let s_t be the sequence of observed timed events associated with p'_i. If the condition presented in Definition 4.2 does not hold, then there exists a sequence of timed events $s'_t \in \overline{\{s_t\}}$, such that a state $x' = f(x_0, \phi(s'_t))$, with $\tilde{\lambda}(x') = y_{z,l_z}$ is reached. Since $\|s'_t\| = l'_z - 1$, then according to lines 24 and 25 of Algorithm 4.1, the model is either reinitialized or stopped after the observation of s'_t, and in this case path p'_i is not necessarily played in the model. Thus, if the condition for reinitializability presented in Definition 4.2 is not satisfied, then Algorithm 4.1 cannot be used for fault detection, since it is not guaranteed that the model will be reinitialized after playing a fault-free path associated with p'_i. Sufficient conditions and a method for verifying the reinitializability of a DAOCT model are presented in MOREIRA and LESAGE [22]. These conditions

Algorithm 4.1: Fault detection algorithm

```
Input: Identified TAOCT model, $\tilde{\lambda}$, $V$, $N$
Output: Fault detection
1  Define the current state of the model $x_c \leftarrow x_0$
2  Define the current path estimate $\theta_{e,c} \leftarrow R$
3  Define the counter of event observations $\eta \leftarrow 0$
4  $\tau_{max} \leftarrow \sup \bigcup_{i \in R} \bigcup_{\sigma' \in \Gamma(x_c)} g(x_c, \sigma', i)$
5  Reset the global clock $c_g$
6  while $c_g \leq \tau_{max}$ do
7      Check the occurrence of an event $\sigma$
8      if $\sigma$ is detected then
9          $\tau \leftarrow c_g(t)$
10         Go to line 12
11 Detect the fault and stop the algorithm
12 $\eta \leftarrow \eta + 1$
13 if $\sigma \notin \Gamma(x_c)$ then
14     Detect the fault and stop the algorithm
15 $\theta_{e,n} \leftarrow \theta_{e,c} \cap \theta(x_c, \sigma, \tau)$
16 if $\theta_{e,n} = \emptyset$ then
17     Detect the fault and stop the algorithm
18 if $|\theta_{e,n}| = 1 \wedge |\theta_{e,c}| > 1$ then
19     Find the pair $(i, \mu_i) \in N$ such that $\theta_{e,n} = \{i\}$
20     if $\mu_i > \eta$ then
21         Detect the fault and stop the algorithm
22 Define state $x_n \leftarrow f(x_c, \sigma)$
23 Define set $\Lambda \leftarrow \{l'_i : i \in \theta_{e,n}\}$
24 if there exists $l'_i \in \Lambda$ s.t. $\eta = l'_i - 1$, $\tilde{\lambda}(x_n) = y_{i,l'_i}$, and $u_{i,l'_i} = u_{i,1}$ then
25     Go to line 1
26 if there exists $l'_i \in \Lambda$ s.t. $\eta = l'_i - 1$, $\tilde{\lambda}(x_n) = y_{i,l'_i}$, and $u_{i,l'_i} \neq u_{i,1}$ then
27     $x_c \leftarrow x_n$
28     $\theta_{e,c} \leftarrow \emptyset$
29     $\tau_{max} \leftarrow \infty$
30     Go to line 5
31 if $\max_{l'_j \in \Lambda} l'_j = \eta + 1$ then
32     Detect the fault and stop the algorithm
33 $\theta'_{e,n} \leftarrow \{i \in \theta_{e,n} : l'_i = \eta + 1\}$
34 $\theta_{e,n} \leftarrow \theta_{e,n} \setminus \theta'_{e,n}$
35 $x_c \leftarrow x_n$
36 $\theta_{e,c} \leftarrow \theta_{e,n}$
37 Go to line 4
```

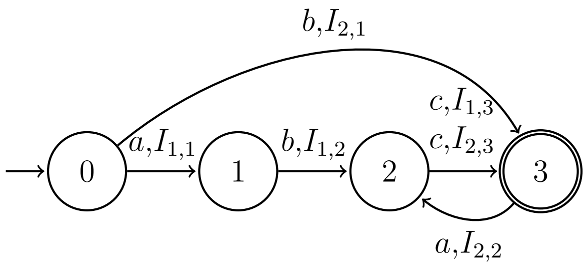


Figure 4.2: TAOCT model of Example 4.1.

and verification method remain valid for the TAOCT, since they are still valid for the underlying DAOCT model.

In the following example, the three situations in which a fault can be detected using the TAOCT, and not using the underlying DAOCT model, are illustrated.

Example 4.1 *Consider the following time-interval paths p'_1 and p'_2:*

$$p'_1 = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a, I_{1,1}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, b, I_{1,2}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, c, I_{1,3}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right),$$

where $I_{1,1} = [25.3, 95.2]$, $I_{1,2} = [0, 47.7] \cup [148.8, 246.4] \cup [505.2, 705.9]$ and $I_{1,3} = [165.9, 481.3] \cup [818.9, 1133.6]$, and

$$p'_2 = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, b, I_{2,1}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, a, I_{2,2}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, c, I_{2,3}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

with $I_{2,1} = [200, 215]$, $I_{2,2} = [90, 110]$ and $I_{2,3} = [590, 620]$.

The TAOCT obtained by applying Algorithm 3.1, for $k = 1$, is shown in Figure 4.2. The set of states of the TAOCT is $X = \{0, 1, 2, 3\}$. Label σ,I on each transition from a state $x \in X$ means that a guard $g(x, \sigma, i) = I$ is defined. The final states X_f are represented by double circles as in Section 2.5.

A fault that leads the fault detector to a deadlock (first type of fault) would be detected using Algorithm 4.1 if, for instance, after the occurrence of sequence $s''_t = (a, 60)(b, 200)$, no event is observed at state 2 after the maximum time $\tau_{max} = 1133.6$ seconds has elapsed.

Suppose now that sequence $s'_t = (a, 60)(b, 200)(c, 100) \in \Sigma_t^\star$ is observed, i.e., event c is observed in state 2 after 100 seconds. In this case, although c is feasible in state 2, since $100 \notin I_{1,3}$, then a fault is detected using Algorithm 4.1. As the time value 100 does not belong to any guard defined in this transition, the detected fault is of the second type. Note that this fault would not be detected if the underlying DAOCT were used.

Suppose that during system operation timed sequence $s_t = (a,60)(b,200)(c,600) \in \Sigma_t^\star$ *is observed. Thus, since event* a *is feasible in state* 0*, and* a *occurs at time instant* $60 \in I_{1,1}$*, then* $(a,60) \in L_{t,Iden}$*, and the path estimation function is* $\theta_e(0,(a,60)) = \{1\}$*. After the occurrence of event* b*, no fault is detected since* b *is feasible in state* 1*,* $200 \in I_{1,2}$*, and* $\theta_e(0,(a,60)(b,200)) = \theta_e(0,(a,60)) \cap \theta(1,b,200) = \{1\} \cap \{1\} = \{1\}$*. However, after the occurrence of event* c *after* 600 *seconds in state* 2*, the fault is detected, since, although* c *is feasible in state* 2*, time instant* $600 \notin I_{1,3}$ *and* $\theta(2,c,600) = \{2\}$*. Thus,* $\theta_e(0,(a,60)(b,200)(c,600)) = \theta_e(0,(a,60)(b,200)) \cap \theta(2,c,600) = \{1\} \cap \{2\} = \emptyset$*. It is important to remark that the fault would not be detected in this example if the underlying DAOCT model were used, since transition* $(2,c,3)$ *belongs to the untimed paths associated with* p_1' *and* p_2'*. Note that the last event observation is consistent with a path that is not one of the estimated paths. Hence, this is a fault of the third type.* □

It is important to remark that all faults that can be detected using only the logical behavior of the system, modeled by the underlying DAOCT, can also be detected using the TAOCT model. Thus, there is an improvement in the fault detection capability using this timed model.

Chapter 5

Fault detection of a sorting unit system

5.1 Closed-loop system behavior

The identification method for obtaining the TAOCT is illustrated using the sorting unit system presented in Figure 5.1. Three different types of pieces are sorted in the system: white plastic pieces (WP), black plastic pieces (BP), and metallic pieces (M). Each type of piece is pushed to one of the three slides shown on the bottom of Figure 5.1, such that pieces of type WP are pushed to the right slide, pieces of type M are pushed to the slide in the middle, and pieces of type BP are pushed to the left slide.

On the right of Figure 5.1, there is a stack magazine where the pieces can be stored in any order. In the sorting process, the pieces at the bottom of the stack magazine are placed onto the conveyor belt by a pneumatic pusher. Then, the conveyor belt is turned on, and the piece is moved in the direction of two sensors in order to determine its type. An inductive sensor detects metallic pieces (type M), and an optical sensor detects metallic (type M) and white plastic pieces (type WP). If a black plastic piece (type BP) is on the conveyor, then none of these two sensors is capable of detecting it. The optical sensor is located close to the inductive sensor, such that metallic pieces are detected by both sensors almost at the same time.

It is also important to remark that there is a photoelectric sensor next to each sorting pusher on the conveyor. When a piece is detected by the photoelectric sensor next to the pusher that should remove it from the conveyor, the conveyor is stopped and the pusher is extended. Then, the pusher is retracted and a new piece can be placed on the conveyor by the pusher of the stack magazine.

The sorting pushers are denoted as SP$_{left}$, SP$_{center}$ and SP$_{right}$, for the left, center and right sorting pusher of Figure 5.1, respectively. The magazine pusher is denoted

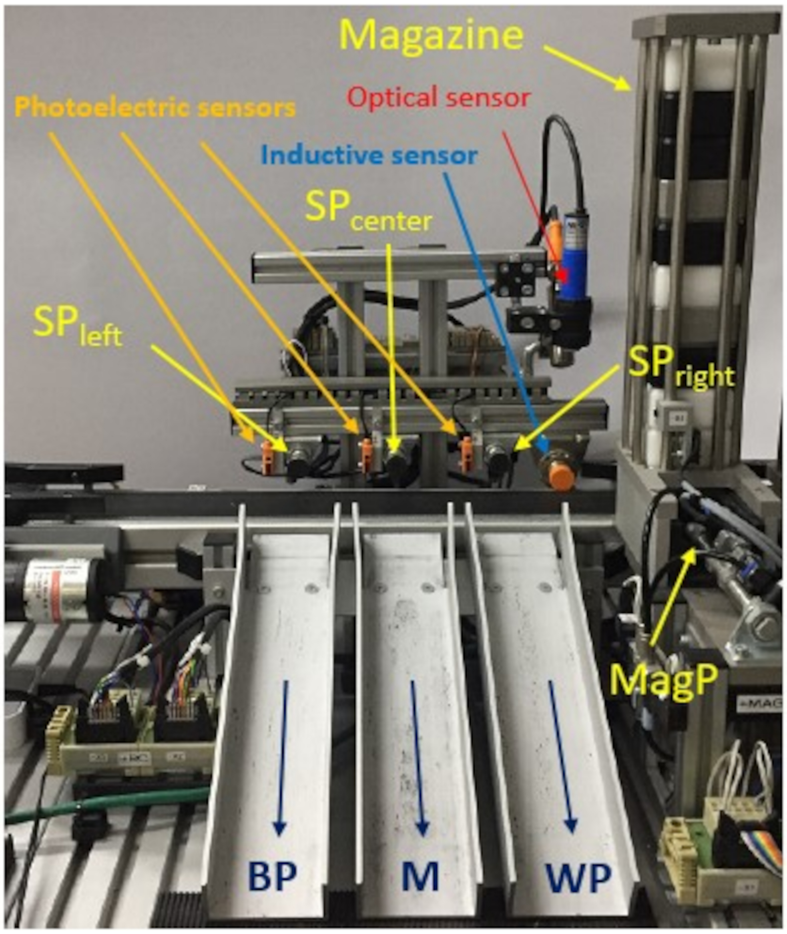


Figure 5.1: Sorting unit system of the practical example.

as MagP. The sorting unit system has 13 sensors and 6 actuator signals. Thus, the controller has 19 input and output signals. Table 5.1 gives the position number d of each controller signal in the I/O vector, along with their description.

The initial state of all observed paths is defined as the I/O vector corresponding to the case where the conveyor belt is turned off, and all pushers are retracted.

5.2 TAOCT model computation

During the identification process, 2294 timed paths with lengths ranging from 6 to 14 I/O vectors were observed, corresponding to two hours and fifty three minutes of observation of the controller signals. A total of 12 logically distinguishable paths were obtained. As there are only three types of pieces, only three different logically distinguishable sequences would be expected. However, it has been observed that, since the inductive and optical sensors are very close to each other, the order of sensor readings (rising and falling edges) may change for different sorting cycles of metallic pieces, increasing the number of paths.

In this practical example, the Nearest Neighbors Method, presented in Section

Table 5.1: Table containing the description of each I/O signal.

d	**Description of the I/O signal**
1	Sensor indicating the extension of MagP
2	Sensor indicating the retraction of MagP
3	Sensor indicating the extension of SP$_{left}$
4	Sensor indicating the retraction of SP$_{left}$
5	Sensor indicating the extension of SP$_{center}$
6	Sensor indicating the retraction of SP$_{center}$
7	Sensor indicating the extension of SP$_{right}$
8	Sensor indicating the retraction of SP$_{right}$
9	Optical sensor
10	Inductive sensor
11	Photoelectric sensor of SP$_{left}$
12	Photoelectric sensor of SP$_{center}$
13	Photoelectric sensor of SP$_{right}$
14	Command to activate SP$_{left}$
15	Command to activate SP$_{center}$
16	Command to activate SP$_{right}$
17	Command to extend MagP
18	Command to retract MagP
19	Command to activate the conveyor belt

3.3.1, has been selected for the computation of the sets $I_{i,j}$ of the time-interval paths. The threshold $\zeta = 80$ms has been chosen, since we consider that this value is consistent with the dynamics of each transition. The uncertainty δ has been considered to be 1ms, which is equal to the measured scan cycle.

After obtaining the time-interval paths p'_i, the TAOCT model is computed following the steps of Algorithm 3.1. The identified TAOCT for $k = 1$ is depicted in Figure 5.2. In this case, the identified TAOCT has 26 states and 38 transitions. In Figure 5.2, the guards are not presented due to lack of space. The events of the TAOCT are rising and falling edges of the elements of the I/O controller vector, which are represented by $\uparrow d$ and $\downarrow d$, respectively, where d is the position of the signal in the I/O vector. It is important to remark that, according to Definition 2.6, an event can be formed of more than one rising or falling edge of controller signals.

The programming code for computing the TAOCT model was implemented using Python 3.7 on a computer Intel Core i5 with 2.4GHz and 4GB RAM. The time required for computing the time-interval paths from the fault-free observed timed paths was 424ms. The most time-consuming operations are those related to the formation of clusters, which are carried out for every transition of every observed path. The time required for the computation of the TAOCT model was only 2.4ms. Thus, the overall time for the computation of the TAOCT model, including the computation of the time-interval paths p'_i, is approximately 426ms.

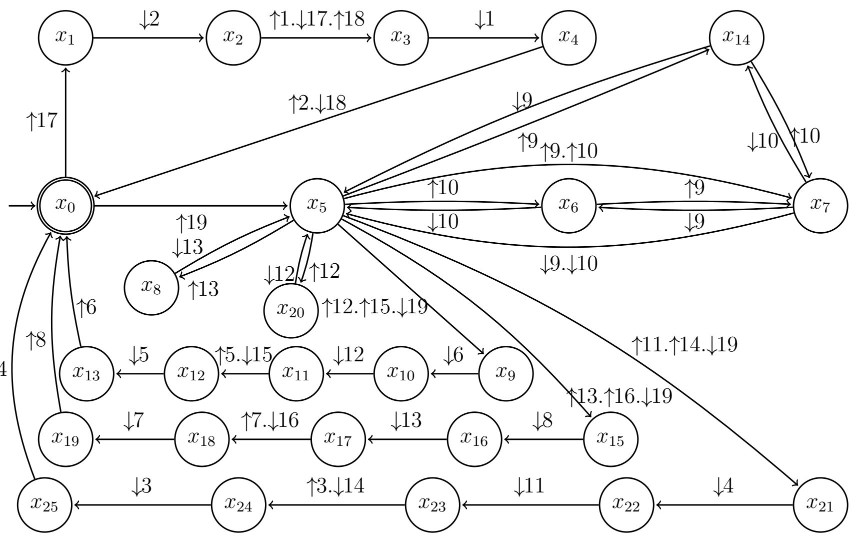


Figure 5.2: TAOCT model obtained for the practical example.

Table 5.2: Computation time of time-interval paths using the clustering method based on K-Means, for each value of K_{max}.

K_{max}	**Time for computing the time-interval paths [s]**
1	0.506
2	27.4
3	36.2
4	44.7
5	54.5
6	63.1
7	72.6
8	85.8

It is worth noting that the use of the clustering method based on K-Means for computing the time-interval paths has proven to be much more resource-expensive than the use of the Nearest Neighbors Method. Table 5.2 shows the times required to obtain the time-interval paths by choosing different values for K_{max}. The selected parameters of the clustering procedure were $\xi = 0.2$ and $\delta = 1$ms.

It can be seen in Table 5.2 that when the K_{max} is chosen to be equal to one, meaning that no clusters are to be formed, the computation time is similar to the one observed using the Nearest Neighbors Method. However, for $K_{max} \geq 2$, the method based on K-Means takes many seconds to compute the time-interval paths. The reason for this loss in performance is that the K-Means must be performed for every value of $K \in \{1, 2, \ldots, K_{max}\}$, which significantly impacts the computation time. Thus, even though the method based on K-Means does not require any knowledge about the dynamics of the transitions, the cost of applying this method may be very high for large systems.

5.3 Fault detection based on the TAOCT model

In the sequel, three faulty scenarios for which the fault can be detected thanks to the timing information added to the TAOCT model are presented.

In the first scenario, consider that, after a piece is placed on the conveyor belt by pusher MagP, the pusher stuck extended and cannot be retracted. The path associated with this behavior is $p = (x_0, \uparrow 17, x_1, \downarrow 2, x_2, \uparrow 1. \downarrow 17. \uparrow 18, x_3)$. In this case, the system deadlocks, since the conveyor belt is turned on only after the retraction of the pusher is detected ($\uparrow 2$), which never occurs. This fault cannot be detected by using the DAOCT model, but can be detected by using the TAOCT model, since the falling edge of the sensor that indicates that MagP is extended ($\downarrow 1$) must occur before the maximum time of the guard $g(x_3, \downarrow 1, 1) = [87, 161]$. Thus, when the elapsed time is greater than 161 milliseconds, the occurrence of a fault is detected.

To illustrate the second faulty scenario, consider a fault in the speed controller of the conveyor that makes it work faster than expected. Consider, for instance, that after a BP piece is placed on the conveyor belt, and the conveyor is turned on ($\uparrow 19$), which corresponds to state x_5 of the TAOCT, it reaches the photoelectric sensor next to the first sorting pusher ($\uparrow 13$) in a time smaller than the minimum time of the guard $g(x_5, \uparrow 13, 4) = [4058, 4146]$. Thus, the fault detector is able to indicate that some fault has occurred. It is important to remark that, since $\uparrow 13$ is coherent with the logical behavior of the system, then the detection system based on the underlying DAOCT model would not be capable of detecting this fault.

The third faulty scenario can be illustrated by the following example. Consider all observed paths associated with BP or M pieces, and consider that the piece is in front of the photoelectric sensor next to the right sorting pusher, *i.e.*, the rising edge of the photoelectric sensor ($\uparrow 13$) has been observed, which corresponds to state x_8 of Figure 5.2. By analyzing the time elapsed between $\uparrow 13$ and $\downarrow 13$, two distinct sets of time values can be defined according to the type of piece, as shown in Figure 5.3. This occurs because metallic pieces are detected for a longer time than plastic pieces by the photoelectric sensor due to their brightness. Consider now a fault that causes both the optical and inductive sensors to fail at the same time. In this case, a piece of type M would lead to the same logical behavior as a BP piece, making such a fault non-detectable by the diagnosis system based on the underlying DAOCT model. However, as shown in Figure 5.3, it is possible to distinguish the types of pieces by using the guards associated with the timed paths. While a BP piece would take some time in the interval $g(x_8, \downarrow 13, 4) = [915, 937]$, a metallic piece would take some value in the interval $[1035, 1052]$ milliseconds, which corresponds to the union of all guards defined in state x_8, for event $\downarrow 13$, and the timed paths associated with metallic pieces. Thus, if a metallic piece is on the conveyor belt,

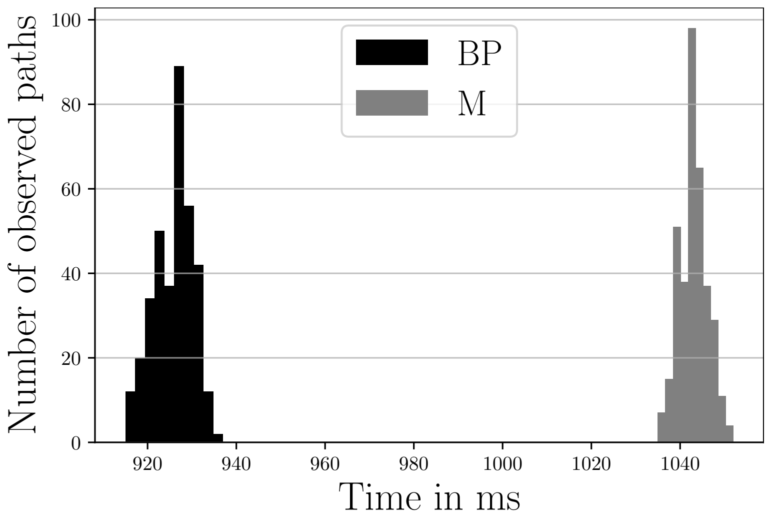


Figure 5.3: Histogram showing the observed times elapsed between the rising edge and falling edge of the photoelectric sensor next to the first sorting pusher (BP in black and M in grey) in the observed timed paths.

and the fault occurs, the time elapsed between the rising and falling edges of the photoelectric sensor will be compatible with the guard condition associated with metallic pieces, and non-compatible with the guard condition of black plastic pieces. Since the logical behavior is not coherent with the timing behavior, the detector determines that a fault has occurred in the system.

Remark 5.1 *In the practical example shown in this chapter, it has been assumed that the fault detector is implemented in the same PLC that was used in the identification procedure. By doing so, it is easier to ensure that the fault detector runs properly, since the scan cycle is the same for both the detector and the system. However, for many applications, it may be more convenient to implement the detector on a different device, such as a computer running the fault detector algorithm written in a programming language such as Python or C. In the case where a different device is used, attention must be paid to issues that may cause the detector to work in an unexpected way. For instance, if the scan cycle of the exterior device is slower than the scan cycle of the PLC, then there is a chance that the detector will miss some event observations, in which case the model no longer tracks the behavior of the system.* □

Chapter 6

Conclusions

In this work, a new timed model for the identification of DES with the aim of fault detection is proposed. In this model, called Timed Automaton with Outputs and Conditional Transitions (TAOCT), timing information regarding the occurrence of events is added in the form of guards to the transitions. The identification procedure for obtaining the TAOCT is thoroughly developed, from the observation of the fault-free behavior of the system until the computation of the model.

In the identification process, time-interval paths are defined by considering sets of time values associated with each transition of the path. The computation of these sets is carried out through the use of a clustering procedure, and each one of the obtained clusters of time values corresponds to a different timing behavior associated with that transition. Two methods that are based on clustering procedures found in the literature are proposed with the aim of computing the required clusters: one is based on the Nearest Neighbors Method and the other one is based on the K-Means clustering method. Examples illustrating the use of both methods are presented and discussed. The separation of the observed timed values in clusters improves the accuracy of the guards defined in the TAOCT in comparison with other timed models in the literature, which usually deal only with guards represented as single intervals.

After the presentation of the identification procedure, the algorithm for computing the TAOCT model is proposed. It is shown that the identified model obtained with this algorithm simulates the observed fault-free behavior of the system, which is an important property for an identified model meant to be used for fault detection purposes.

The algorithm for fault detection using the identified TAOCT model has also been proposed, and the different scenarios in which a fault can be detected thanks to the timing information added to the TAOCT are presented. In fact, a refinement of the path estimation can be carried out using the timing information, and thus faults that cannot be detected by using untimed models can be detected by using the TAOCT.

The proposed method has been applied in a practical example, where the fault-free behavior of a didactic manufacturing plant is observed and the recorded data is used to obtain the TAOCT model. The practical use of the computed model in a fault detection scheme has proven to be successful in detecting faults that would not have been detected by using other identified models.

An important improvement of this work making the presented approach more suitable for an industrial environment would be the proposal of a mechanism allowing the identified model to be constantly updated while the system is running. For example, if the occurrence of a false alarm is detected, then its corresponding sequence can be added to the observed behavior and the model could be recomputed taking into account the newly added information. Alternatively, if the occurrence of a fault is missed by the detector, then one or more parameters used for identification (k, ζ, K_{max}, etc.) could be changed accordingly, as a way of improving the identified fault-free model. The implementation of an automated procedure through which the model is updated, using artificial intelligence for instance, would be a very interesting option for further research and certainly a leap forward in the industrial application of the results presented in this work.

For industrial purposes, it would also be essential to expand the ideas developed in this work to include the possibility of executing several system paths concurrently in the model. Indeed, in the practical example presented in Chapter 5, only one path is executed by the system at any given time, since the pieces are processed one at a time, with the next piece only arriving once the current cycle is complete. Allowing the possibility of concurrent paths is very important, since they often occur in real industrial systems. In the practical example of Chapter 5, it would be possible to address the case where a piece is introduced in the system before the completion of the production cycle corresponding to the piece that came before.

Another possible direction for future research could be to generalize the TAOCT model to the case where the passage from one discrete state of the model to another one may also depend on the evolution of multiple continuous variables. Indeed, in the case of the TAOCT, the only continuous variable on which the discrete behavior depends is the global clock. However, the inclusion of other variables such as temperature or pressure may also be envisioned as a way of refining even further the path estimation function and improving fault detection in systems where measurements of such variables are available.

References

[1] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of Discrete-Event Systems", *IEEE Transactions on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.

[2] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "Coordinated Decentralized Protocols for Failure Diagnosis of Discrete Event Systems", *Discrete Event Dynamic Systems*, v. 10, n. 1-2, pp. 33–86, 2000.

[3] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems", *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1679–1684, 2011.

[4] ZAYTOON, J., LAFORTUNE, S. "Overview of fault diagnosis methods for discrete event systems", *Annual Reviews in Control*, v. 37, n. 2, pp. 308–320, 2013.

[5] SANTORO, L. P. M., MOREIRA, M. V., BASILIO, J. C. "Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers", *Automatica*, v. 77, pp. 93–102, 2017.

[6] CABRAL, F. G., MOREIRA, M. V. "Synchronous Diagnosis of Discrete-Event Systems", *IEEE Transactions on Automation Science and Engineering*, v. 17, n. 2, pp. 921–932, 2020.

[7] CHEN, Y.-L., PROVAN, G. "Modeling and diagnosis of timed discrete event systems - a factory automation example". In: *Proceedings of the 1997 American Control Conference*, pp. 31–36, Albuquerque, NM, USA, 1997. IEEE.

[8] PANDALAI, D. N., HOLLOWAY, L. E. "Template languages for fault monitoring of timed discrete event processes", *IEEE Transactions on Automatic Control*, v. 45, n. 5, pp. 868–882, 2000.

[9] TRIPAKIS, S. “Fault diagnosis for timed automata”. In: *International symposium on formal techniques in real-time and fault-tolerant systems*, pp. 205–221, Oldenburg, Germany, 2002. Springer.

[10] ZAD, S. H., KWONG, R. H., WONHAM, W. M. “Fault diagnosis in discrete-event systems: Incorporating timing information”, *IEEE Transactions on Automatic Control*, v. 50, n. 7, pp. 1010–1015, 2005.

[11] VIANA, G. S., MOREIRA, M. V., BASILIO, J. C. “Codiagnosability analysis of discrete-event systems modeled by weighted automata”, *IEEE Transactions on Automatic Control*, v. 64, n. 10, pp. 4361–4368, 2019.

[12] MEDHI, S. O. E., LECLERCQ, E., LEFEBVRE, D. “Petri nets design and identification for the diagnosis of discrete event systems”. In: *IAR Annu. Meeting*. IEEE, 2006.

[13] CABASINO, M. P., GIUA, A., SEATZU, C. “Identification of Petri nets from knowledge of their language”, *Discrete Event Dynamic Systems*, v. 17, n. 4, pp. 447–474, 2007.

[14] DOTOLI, M., FANTI, M. P., MANGINI, A. M. “Real time identification of discrete event systems using Petri nets”, *Automatica*, v. 44, n. 5, pp. 1209–1219, 2008.

[15] ESTRADA-VARGAS, A. P., LOPEZ-MELLADO, E., LESAGE, J.-J. “A comparative analysis of recent identification approaches for discrete-event systems”, *Mathematical Problems in Engineering*, v. 2010, pp. 453254, 2010.

[16] DOTOLI, M., FANTI, M. P., MANGINI, A. M., et al. “Identification of the unobservable behaviour of industrial automation systems by Petri nets”, *Control Engineering Practice*, v. 19, n. 9, pp. 958–966, 2011.

[17] CABASINO, M. P., GIUA, A., HADJICOSTIS, C. N., et al. “Fault model identification and synthesis in Petri nets”, *Discrete Event Dynamic Systems*, v. 25, n. 3, pp. 419–440, 2015.

[18] ESTRADA-VARGAS, A. P., LÓPEZ-MELLADO, E., LESAGE, J. “A Black-Box Identification Method for Automated Discrete-Event Systems”, *IEEE Transactions on Automation Science and Engineering*, v. 14, n. 3, pp. 1321–1336, 2017.

[19] KLEIN, S., LITZ, L., LESAGE, J.-J. “Fault detection of Discrete Event Systems using an identification approach”, *IFAC Proceedings Volumes*, v. 38, n. 1, pp. 92–97, 2005. 16th IFAC World Congress.

[20] ROTH, M., LESAGE, J.-J., LITZ, L. "An FDI method for manufacturing systems based on an identified model", *IFAC Proceedings Volumes*, v. 42, n. 4, pp. 1406–1411, 2009. 13th IFAC Symposium on Information Control Problems in Manufacturing.

[21] MOREIRA, M. V., LESAGE, J.-J. "Discrete event system identification with the aim of fault detection", *Discrete Event Dynamic Systems*, v. 29, n. 2, pp. 191–209, 2019.

[22] MOREIRA, M. V., LESAGE, J.-J. "Fault diagnosis based on identified discrete-event models", *Control Engineering Practice*, v. 91, pp. 104101, 2019.

[23] ROTH, M., LESAGE, J.-J., LITZ, L. "The concept of residuals for fault localization in discrete event systems", *Control Engineering Practice*, v. 19, n. 9, pp. 978–988, 2011.

[24] DE SOUZA, R. P. C., MOREIRA, M. V., LESAGE, J.-J. "A hierarchical approach for discrete-event model identification incorporating expert knowledge". In: *15th IFAC Workshop On Discrete Event Systems (WODES'20)*, pp. 275–281, Rio de Janeiro, Brazil, 2020.

[25] BASILE, F., CHIACCHIO, P., COPPOLA, J., et al. "Identification of Petri nets using timing information". In: *3rd International Workshop on Dependable Control of Discrete Systems (DCDS2011)*, pp. 154–161, Saarbrücken, Germany, 2011. IEEE.

[26] BASILE, F., CHIACCHIO, P., COPPOLA, J. "A novel model repair approach of timed discrete-event systems with anomalies", *IEEE Transactions on Automation Science and Engineering*, v. 13, n. 4, pp. 1541–1556, 2016.

[27] BASILE, F., CHIACCHIO, P., COPPOLA, J. "Identification of time Petri net models", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 47, n. 9, pp. 2586–2600, 2017.

[28] SUPAVATANAKUL, P., LUNZE, J., PUIG, V., et al. "Diagnosis of timed automata: Theory and application to the DAMADICS actuator benchmark problem", *Control Engineering Practice*, v. 14, n. 6, pp. 609–619, 2006.

[29] SCHNEIDER, S. *Automatic Modeling and Fault Diagnosis of Timed Concurrent Discrete Event Systems.* Ph.D. thesis, École Normale Supérieure de Cachan - ENS Cachan, 2015.

[30] DAS, S. R., HOLLOWAY, L. E. "Characterizing a confidence space for discrete event timings for fault monitoring using discrete sensing and actuation

signals", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, v. 30, n. 1, pp. 52–66, 2000.

[31] SCHNEIDER, S., LITZ, L., LESAGE, J.-J. "Determination of Timed Transitions in Identified Discrete-Event Models for Fault Detection". In: *51st IEEE Annual Conference on Decision and Control (CDC'12)*, pp. 5816–5821, Maui, HI, USA, 2012.

[32] DE SOUZA, R. P. C., MOREIRA, M. V., LESAGE, J.-J. "A timed model for discrete event system identification and fault detection". In: *21st IFAC World Congress*, pp. 826–831, Berlin, Germany, 2020.

[33] DE SOUZA, R. P. C., MOREIRA, M. V., LESAGE, J.-J. "Fault detection of Discrete-Event Systems based on an identified timed model", *Control Engineering Practice*, v. 105, pp. 104638, 2020.

[34] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2nd ed. New York, NY, USA, Springer Science+Business Media, LLC, 2008.

[35] ALUR, R., DILL, D. L. "A theory of timed automata", *Theoretical computer science*, v. 126, n. 2, pp. 183–235, 1994.

[36] HARTIGAN, J. *Clustering algorithms*. New York, NY, USA, John Wiley & Sons, Inc., 1975.

[37] LU, S.-Y., FU, K. S. "A sentence-to-sentence clustering procedure for pattern analysis", *IEEE Transactions on Systems, Man, and Cybernetics*, v. 8, n. 5, pp. 381–389, 1978.

[38] JAIN, A. K., MURTY, M. N., FLYNN, P. J. "Data Clustering: A Review", *ACM Comput. Surv.*, v. 31, n. 3, pp. 264–323, 1999.

[39] PHAM, D. T., DIMOV, S. S., NGUYEN, C. D. "Selection of K in K-means clustering", *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, v. 219, n. 1, pp. 103–119, 2005.

[40] SUGAR, C. A., JAMES, G. M. "Finding the number of clusters in a dataset: An information-theoretic approach", *Journal of the American Statistical Association*, v. 98, n. 463, pp. 750–763, 2003.