Contents lists available at ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Experimental evaluation of an active fault–tolerant control method

M. Schuh[*], M. Zgorzelski, J. Lunze

[a] Institute of Automation and Computer Control, Ruhr-Universität Bochum, Germany

ARTICLE INFO

Article history:
Received 21 July 2014
Accepted 3 June 2015
Available online 1 July 2015

Keywords:
Fault tolerance
Discrete-event systems
I/O automata
Diagnosis
Control design

ABSTRACT

A method for the active fault–tolerant control of systems modeled by deterministic input/output (I/O) automata is presented and evaluated experimentally. In the fault-free case, a given controller moves the system into a specified final state. The aim of the paper is to construct a framework which guarantees that the final state is reached again after the occurrence of a fault. In this paper, two existing methods, one for the fault diagnosis and one for the reconfiguration of I/O automata, are combined in order to obtain an active and completely autonomous FTC framework. It is shown how the developed method can be applied to a handling system and that it indeed makes the faulty system again fulfill its given task.

1. Introduction

This paper deals with active fault–tolerant control (FTC) of discrete event systems modeled by deterministic input/output (I/O) automata. As long as no fault is present, the plant $\mathcal{P}$ is controlled by the nominal controller $\mathcal{C}$ such that a given specification $\mathcal{S}$, for example the achievement of a final state, is fulfilled (Fig. 1). When a fault occurs, the behavior of the plant is affected such that the nominal controller cannot steer the plant according to the specification any longer. The aim of active FTC is to modify the controller $\mathcal{C}$ such that the faulty plant again adheres to the specification. After the occurrence of a fault, two main steps have to be performed autonomously by the active FTC method:

1. Fault detection and identification.
2. Reconfiguration of the controller.

To accomplish these steps, an active FTC loop as shown in Fig. 1 is considered in this paper. The active FTC loop consists of two layers:

- an execution layer, where the plant $\mathcal{P}$ and the controller $\mathcal{C}$ form a classical feedback control loop,
- a supervision layer containing the diagnostic unit $\mathcal{D}$ and the reconfiguration unit $\mathcal{R}$.

In previous publications, the two steps of diagnosis and reconfiguration have been considered independently. In Schmidt and Lunze (2013a), a method for the active diagnosis of deterministic I/O automata has been presented (cf. right part of Fig. 1), whereas Nke and Lunze (2011b) deals with the online reconfiguration for this system class (cf. left part of Fig. 1). Even though both methods consider the same system class, three main steps have to be undertaken in order to combine them to an active and completely autonomous FTC framework:

1. The diagnostic unit $\mathcal{D}$ has to be extended such as to provide exactly the information that the reconfiguration unit $\mathcal{R}$ requires.
2. Both existing methods have to be modified as some previously made assumptions (e.g. on the instantaneous identification of the fault) are no longer valid.
3. A component managing the switching among inputs from the controller, the diagnostic unit and the reconfiguration unit has to be introduced.

Literature review: There are several approaches for the FTC of discrete event systems, most of which use standard automata and the Supervisory Control Theory (SCT) and fall into three categories:

1. Robust or passive fault–tolerant controllers.
2. Switching controllers.
3. Controllers with checkpoint states.

To the best of our knowledge, no other approach for the fault–tolerant control of discrete event systems modeled by I/O automata as considered in this paper exists. This system class is especially suited for modeling dynamic systems in which the inputs from a controller and the outputs of the plant form an explicit causality relation. While the supervisors resulting from

[*] Corresponding author. Fax: +49 2343214101.
E-mail addresses: melanieschuh@atp.rub.de (M. Schuh), zgorzelski@atp.rub.de (M. Zgorzelski), lunze@atp.rub.de (J. Lunze).

http://dx.doi.org/10.1016/j.conengprac.2015.06.001
0967-0661/© 2015 Elsevier Ltd. All rights reserved.

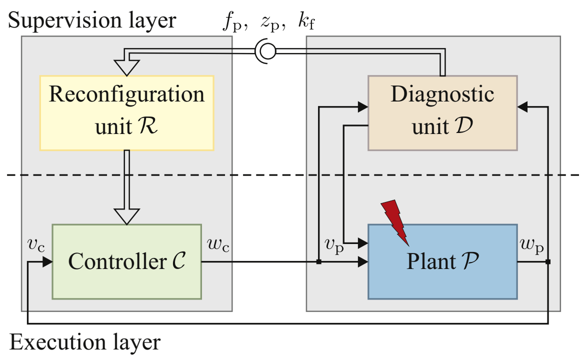

Fig. 1. Active FTC loop.

SCT are designed such as to *prevent* the system from executing forbidden motions, the control aim in this paper is to *explicitly steer* the system into a desired final state.

In the first category of literature mentioned above, a single controller is designed which guarantees the fulfillment of the nominal specification (Park & Lim, 1999; Saboori & Hashtrudi Zad, 2006) or a degraded specification (Wen, Kumar, Huang, & Liu, 2008; Wittmann, Richter, & Moor, 2012) even when a fault occurred. Therefore, no explicit diagnostic result is necessary. In contrast to this, the method presented in this paper is an *active* FTC method for which the present fault has to be identified.

In the second category, a bank of controllers corresponding to different fault cases (Darabi, Jafari, & Buczak, 2003; Shu & Lin, 2014) or different system states at the time the fault occurs (Paoli & Sartini, 2011) is designed offline. Then, once a fault is identified, the respective controller is connected to the plant. On the contrary, in this paper only one nominal controller is designed, which is adapted online to the actual fault. Therefore, there is no need to precompute and store a controller for every possible fault.

In the third category, certain restart states within the controller are selected, to which the system is returned after a fault occurred and from which the process can be restarted afterwards (Andersson, Lennartson, & Fabian, 2009). It is assumed that methods for the fault diagnosis and the physical transfer of the system into the restart state are given. In this paper, no explicit restart states are considered, but rather the system is always returned to a state in its nominal state sequence.

In summary the FTC method in this paper fundamentally differs from existing methods, because a different system class (I/O automata instead of standard automata) and control aim (reaching final state instead of preventing forbidden behavior) is considered and hence a completely different approach is necessary to solve the FTC problem. A preliminary version of this paper has been published as Schmidt and Lunze (2014). This work is extended here to include a discussion of the practical relevance and applicability of the presented method by evaluating an experiment on a handling system.

Main contribution: The main contributions of the paper consist of the validation of the active FTC framework presented in Schmidt and Lunze (2014), where the methods of active fault diagnosis (Schmidt & Lunze, 2013a) and online reconfiguration (Nke & Lunze, 2011b) are combined, through its application on an experimental setup. It is shown that the experimental evaluation confirms the theoretical results of the above-mentioned work.

Organization of the paper: The FTC problem to be solved is presented in Section 2, where the system model is introduced, the FTC problem is formalized and the methods to be combined including necessary modifications are discussed. Section 3 contains the main theoretical result of the paper and describes the proposed FTC framework in detail. The setup for the experimental evaluation of the developed method is described in Section 4. Section 5 presents the design of the fault-tolerant controller for the considered handling system, while in Section 6 the results of the experimental evaluation are discussed.

2. Problem setup

2.1. Deterministic I/O automata

The plant $\mathcal{P}$ under consideration is modeled by a set $\{\mathcal{A}_f : f \in \mathcal{F} \cup \{0\}\}$ of *deterministic I/O automata*

$$\mathcal{A}_f = (\mathcal{Z}_f, \mathcal{V}, \mathcal{W}, G_f, H_f, z_{f0}) \tag{1}$$

with

- $\mathcal{Z}_f$ – state set,
- $\mathcal{V}$ – set of input symbols (e.g. "turn deflector", "open valve"),
- $\mathcal{W}$ – set of output symbols (e.g. "rising edge at limit switch"),
- $G_f : \mathcal{Z}_f \times \mathcal{V} \rightarrow \mathcal{Z}_f$ – state transition function,
- $H_f : \mathcal{Z}_f \times \mathcal{V} \rightarrow \mathcal{W}$ – output function,
- z_{f0} – initial state,

where $\mathcal{F}$ represents the set of all fault candidates. For $f \in \mathcal{F}$, the automaton $\mathcal{A}_f$ models the behavior of the system when the fault f is present, whereas the automaton $\mathcal{A}_0$ describes the behavior of the fault-free case. It is assumed that all possible faults are known and all models are exact. Only persistent faults are considered, such that the models $\mathcal{A}_f (f \in \mathcal{F})$ do not change over time.

The state transition function

$$G_f(z(k), v(k)) = z(k+1) =: z'(k) \tag{2}$$

maps the current state $z(k)$ and the input $v(k)$ of the system to the next state $z'(k) = z(k+1)$, while the output function

$$H_f(z(k), v(k)) = w(k) \tag{3}$$

of the system yields the corresponding output $w(k)$. Faults are assumed to change the state transition function G_f and the output function H_f compared to the faultless system $\mathcal{A}_0$, but neither its input set $\mathcal{V}$ nor its output set $\mathcal{W}$. The transition from the model $\mathcal{A}_0$ of the faultless system to any model $\mathcal{A}_f (f \in \mathcal{F})$ of the faulty system (e.g. the occurrence of a fault) is not modeled explicitly in this framework.

The state transition function G_f and the output function H_f can be combined to the behavioral relation $L_f : \mathcal{Z}_f \times \mathcal{W} \times \mathcal{Z}_f \times \mathcal{V} \rightarrow \{0, 1\}$, where

$$L_f(z', w, z, v) = \begin{cases} 1 & \text{if } G_f(z, v) = z' \wedge H_f(z, v) = w \\ 0 & \text{otherwise.} \end{cases}$$

For deterministic I/O automata, the behavioral relation L_f is equivalent to a description using the state transition function G_f and the output function H_f.

The *active input function*

$$\mathcal{V}_{af}(z', z) = \{v \in \mathcal{V}_f : G_f(z, v) = z'\} \tag{4}$$

of an automaton $\mathcal{A}_f$ computes all inputs v leading from state z to state z', while the *active output function*

$$\mathcal{W}_{af}(z', z) = \{w \in \mathcal{W}_f : H_f(z, v) = w, v \in \mathcal{V}_{af}(z', z)\} \tag{5}$$

computes the corresponding outputs w.

A deterministic I/O automaton also can be modeled as a graph, where the nodes of the graph represent the states of the automaton and the edges of the graph labeled with the corresponding input/output pairs represent the transitions of the automaton. Due to this fact, some basic notions from graph theory can also be applied to I/O automata. A state $z \in \mathcal{Z}_f$ is called *reachable* from the initial state z_{f0}, if there exists a path in the