Supervisory Control-Based Navigation Architecture: A New Framework for Autonomous Robots in Industry 4.0 Environments

Antonio G. C. Gonzalez, Marcos V. S. Alves, Gustavo S. Viana, Lilian K. Carvalho, *Member, IEEE*, and João C. Basilio, *Senior Member, IEEE*

***Abstract*—Industry 4.0 is characterized by an increasing dependence on automation and interconnection of systems due to the need for more efficient, autonomous, and customizable processes, and so, mobile robot navigation becomes an important tool. In this paper, we present a general methodology for mobile robot navigation in industrial environments in which the open-loop behavior of the robot and the specifications are based on automata. We build a modular supervisor, which is the conjunction of two supervisors: the first one that enforces the robot to follow the path defined by a planner and the second one that guarantees the satisfaction of the specifications such as prevention of collisions and task and movement management. The proposed navigation architecture allows decentralized implementation, in which the modular supervisor is embedded in the mobile robot, whereas the planner runs in an external agent. Such a feature makes the adaptation of the proposed navigation architecture to different environments easy. The navigation architecture proposed in this paper is illustrated by means of a simulation in a hypothetical environment that resembles a smart factory.**

***Index Terms*—Discrete event systems (DES), industry 4.0, mobile robots, supervisory control.**

I. Introduction

IN RECENT years, new challenges to make production processes more efficient, autonomous, and customizable have led to a new industrial revolution. A new concept of industry, called Industry 4.0 [1], has emerged and is currently adopted to denominate the current trend of automation and data exchange in manufacturing technologies by creating a "smart factory" [2]. The fundamentals of Industry 4.0 are cyber-physical systems [3], Internet of things [1], [4], cloud computing, big data [5], and mobile robots [6].

Manuscript received June 13, 2017; revised September 21, 2017, December 11, 2017, and December 21, 2017; accepted December 22, 2017. Date of publication December 29, 2017; date of current version April 3, 2018. This work was supported in part by the Brazilian Research Council (CNPq) under Grant 310980/2013-5 and Grant 462307/2014-0 and in part by the Carlos Chagas Foundation (FAPERJ) under Grant 204872. Paper no. TII-17-1253. *(Corresponding author: João Carlos Basilio.)*

The authors are with the Department of Electrical Engineering, Universidade Federal do Rio de Janeiro, Rio de Janeiro 21949-900, Brazil (e-mail: angacego@poli.ufrj.br; mvalves@poli.ufrj.br; gustavo.viana@poli.ufrj.br; lilian.carvalho@poli.ufrj.br; basilio@poli.ufrj.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2017.2788079

In this paper, we consider factories with smart machines that are part of a distributed production line that requires a mobile robot in order to carry the parts they produce either to store them or to take them to another machine for further processing. The robot is connected remotely to computer systems. Our work focuses on providing intelligence to the mobile robot so as it correctly establishes the connection between the machines. To this end, we propose a navigation architecture based on modular supervisory control that, besides establishing the connection between the machines, it also performs the supervision of the robot navigation.

Autonomous mobile robot navigation consists of four stages: mapping, localization, planning, and execution [7]. We consider industrial environments where the structure rarely undergoes major modification, and thus it is reasonable to assume that the environment map is known *a priori*. Since there exist several techniques to deal with the robot localization, we will not address this issue in this paper, focusing only on planning and execution. Finally, regarding the navigation architecture classification usually deployed in the literature [8], we mention that the architecture used in this paper can be regarded as hybrid: being predominantly deliberative with reactive elements to deal with obstacles and sudden changes in the environment where robot navigation takes place.

A. Objective of the Paper

The objective of this paper is to propose a supervisory control approach for mobile robot navigation in industrial environments, such as warehouses and smart factories. We model the environment, the planning structure, and the robot as automata and use modular supervisory control theory [9] to develop a navigation system for mobile robots. The modular supervisory controller ensures the correct navigation of the robot in the presence of unpredictable obstacles and is obtained by the conjunction of two supervisors: a first one that enforces the robot to follow the path defined by the planner and a second one that imposes other specifications such as prevention of collisions, task and movement management, and distinction between permanent and intermittent obstacles. The idea is to develop a general approach that allows the implementation of specifications by means of modules that depend on the task the robot will perform and on the industrial environment. As will become clear, when the

environment is changed, the only changes that are needed to design a new supervisor are those related to the new event set associated with the new environment.

B. Related Works

Discrete event systems (DES) are systems whose behavior is determined by the asynchronous occurrence of certain event types [10]. Several important practical problems have been addressed using DES theory, ranging from theoretical issues, such as fault diagnosis [11] and supervisory control theory [12], to real systems applications [13]–[15]. The DES formalism has also been proved suitable to deal with mobile robot navigation [16]–[22]. In [16], a DES-based supervisory controller that ensures collision and deadlock avoidance for a group of robots that work in order to concurrently accomplish their missions in a two-dimensional space is proposed. However, it is not clear in [16] how to plan and execute the movements necessary to reach the goal location. In [17], the path planning problem for decentralized systems—not necessarily robot path planning—with action costs was addressed by using several weighted automata. Although such an approach could be adapted to compute robot trajectories for the case of several cooperative robots, it is not suitable when only one robot is being used, since the parameters are assumed to be distributed in [17], which is not the case in our work. In [18], a formal method based on linear temporal logic (LTL) has been employed to describe and model specifications in mobile robot navigation. However, the disadvantage of this approach is the computational effort to convert such logics into Büchi automata, which is not required in our work. In [19], Iqbal *et al.* outlined an integration between graph theory, automata, and Z notation in order to propose a supervisory control design framework for robot navigation systems. Structures for representing the environment and some specifications were presented in [19], but further investigation is still needed in order to use the proposed approach to trajectory planning and execution. In [20], automaton-based models for mobile robot navigation were used to show the viability of using DES theory in the modeling, analysis, and synthesis of behaviors applied to the navigation of a mobile robot using visually guided navigation in an unknown or partially known environment. Since the focus was on execution rather than on planning, the optimal path to reach the goal location was not computed in [20]. A formal synthesis of supervisory control software for multiple robot systems was developed in [21] and subsequently in [22], where scalability was improved. Since the main goal was to manage task planning, the problem considered in [21] and [22] is different from the one addressed here.

C. Structure of the Paper

In Section II, we review some basic concepts on DES. In Section III, we present automaton-based models for the environment, planning structures, and robot motion. In Section IV, we present a formulation for the robot navigation problem, and propose a navigation architecture that is based on modular supervisory control, and, in Section V, we carry out a performance analysis of the algorithm proposed here. In Section VI, we illustrate the paper results by means of simulation carried out using MobileSim version 0.7.5, that can be directly inserted in real platforms such as Pioneer P3DX. Finally, we draw some conclusion and outline future research works in Section VII.

II. Background Preliminaries

A. Discrete Event Systems

DES are dynamic systems with discrete state space whose state evolution is entirely determined by the occurrence of asynchronous events over time [10]. In this paper, we adopt automata as the model formalism to describe DES behavior, and, to this end, we present now a brief review of automaton theory. Readers not familiar with automaton theory are referred to [10] for further details.

Let $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ denote a deterministic finite automaton that models a DES, where X is the finite set of states, Σ is the finite set of events, $f : X \times \Sigma \to X$ is the transition function, partially defined over its domain, function $\Gamma : X \to 2^{\Sigma}$ (where 2^{Σ} denotes the power set of Σ) determines the set of active events of the states of G, i.e., $\Gamma(x) = \{\sigma \in \Sigma : f(x, \sigma) \text{ is defined}\}$, x_0 is the initial state, and X_m is the set of marked states. Throughout the text, Σ^* denotes the Kleene closure of Σ, which is the set of all finite strings formed by events in Σ including the empty string, denoted by ε. The transition function f is extended to $f : X \times \Sigma^* \to X$ in the following manner: $f(x, \varepsilon) = x$ and $f(x, s\sigma) = f(f(x, s), \sigma)$, for all $x \in X$, $s \in \Sigma^*$ and $\sigma \in \Sigma$. The languages generated and marked by G are, respectively, $L(G) = \{s \in \Sigma^* : f(x_0, s) \text{ is defined}\}$ and $L_m(G) = \{s \in L(G) : f(x_0, s) \in X_m\}$. The prefix-closure of a language L is defined as $\overline{L} = \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}$, and, if $L = \overline{L}$, L is said to be prefix-closed.

Let Σ_s and Σ_l be two sets of events such that $\Sigma_s \subseteq \Sigma_l$. The natural projection [12] from Σ_l^* to Σ_s^* is a function that removes from strings in Σ_l^* those events that are not in Σ_s, and is denoted as $P : \Sigma_l^* \to \Sigma_s^*$. The inverse projection P^{-1} is the function that recovers those strings in Σ_l^* that generate a given projection in Σ_s^*, being formally defined as $P^{-1}(t) = \{s \in \Sigma_l^* : P(s) = t\}$. The projection and the inverse projection operations are both extended to languages by applying P and P^{-1} to all strings in the language.

The accessible (or reachable) part of automaton G, denoted by $\text{Ac}(G)$, is the automaton obtained by removing from G all states that are not reachable from the initial state and those transitions that either initiate or end at the removed states. The coaccessible (or coreachable) part of automaton G, denoted by $\text{CoAc}(G)$, is the automaton obtained by removing from G all states from which it is not possible to reach some marked state and their respective transitions. The parallel composition [10] between two automata G_1 and G_2 is denoted as $G_1 \| G_2$, and is performed to synchronize the behaviors of two automata by allowing common events to occur only when they are in the active event sets of the current states of both automata and private events, i.e., events that belong to only one of the automata, to freely occur. The languages generated and marked by $G_1 \| G_2$ are, respectively, $L(G_1 \| G_2) = P_1^{-1}[L(G_1)] \cap P_2^{-1}[L(G_2)]$ and $L_m(G_1 \| G_2) =$

$P_1^{-1}[L_m(G_1)] \cap P_2^{-1}[L_m(G_2)]$, where $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$, for $i = 1, 2$.

B. Supervisory Control of DES

Let $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ be the automaton that models the "uncontrolled behavior" of a DES. In some cases, this behavior is unsatisfactory and needs to be "modified" by some control action. The task of restricting the behavior of a DES can be done by using a structure called supervisor. A supervisory control system is a feedback control system that, through the action of a supervisor S acts by enabling (or disabling) event occurrences based on the strings generated by plant G so as to make the closed-loop behavior equal to a given applicable language requirement $K \subseteq L(G)$. In that case, we say that the closed-loop system S/G (read S controlling G) is such that $L(S/G) = K$.

The supervisor is a mapping $S : L(G) \to 2^{\Sigma}$, where, for a given string $s \in L(G)$, $\Gamma[f(x_0, s)] \cap S(s)$ are the events that are enabled by S at state $f(x_0, s)$. The set of events Σ may be partitioned as $\Sigma = \Sigma_c \dot\cup \Sigma_{\text{uc}}$, where Σ_c and Σ_{uc} are, respectively, the sets of controllable events, which can be disabled by the supervisor, and uncontrollable events, whose occurrence cannot be preempted by the supervisor. In this regard, we say that a supervisor is admissible if for all $s \in L(G)$, $\Gamma[f(x_0, s)] \cap \Sigma_{\text{uc}} \subseteq S(s)$. The language achieved by the closed-loop system S/G is, therefore, recursively characterized as $\varepsilon \in L(S/G)$, and $s\sigma \in L(S/G) \Leftrightarrow s \in L(S/G) \wedge s\sigma \in L(G) \wedge \sigma \in S(s)$. The language marked by the closed-loop system is equal to the part of $L_m(G)$ that remains after the supervisory control action, i.e., $L_m(S/G) = L(S/G) \cap L_m(G)$. A supervisor is said to be nonblocking if $L(S/G) = \overline{L_m(S/G)}$.

A realization of a supervisor S is an automaton $R = (X_r, \Sigma, f_r, \Gamma_r, x_{0_r}, X_r)$ whose states are all marked, and is such that, for all $s \in L(G)$, $S(s) = \Gamma_r[f_r(x_{0_r}, s)]$. Thus, the closed-loop behavior is characterized by $R \| G$, since $L(S/G) = L(R\|G)$ and $L_m(S/G) = L_m(R\|G)$.

A language $K \subseteq L(G)$ is controllable with respect to $L(G)$ and uc if $\overline{K}\Sigma_{\text{uc}} \cap L(G) \subseteq \overline{K}$, and it is said to be closed with respect to $L_m(G)$ (or $L_m(G)$-closed) if $K = \overline{K} \cap L_m(G)$. For a given language $K \subseteq L_m(G)$, there exists a nonblocking supervisor S such that $L_m(S/G) = K$ if, and only if, K is controllable and $L_m(G)$-closed.

The applicable language requirement K is frequently given as the intersection of two (or more) elementary specification languages, i.e., $K = K_1 \cap K_2$, where $K_1, K_2 \subseteq L_m(G)$. In this case, the state space of the automaton that marks K is, in the worst case, equal to the Cartesian product of the state spaces of automata that mark K_1 and K_2. In order to circumvent the drawback of computing K, we can investigate the possibility of synthesizing nonblocking supervisors S_1 and S_2 such that $L_m(S_1/G) = K_1$ and $L_m(S_2/G) = K_2$, and thus, construct the modular control architecture where modular supervisor $S_1 \wedge S_2$ is obtained by computing the conjunction of S_1 and S_2, i.e., for all $s \in L(G)$, $S_1 \wedge S_2(s) = S_1(s) \cap S_2(s)$. As a consequence, $L_m(S_1 \wedge S_2/G) = K_1 \cap K_2$. Moreover, supervisor $S_1 \wedge S_2$ is nonblocking if, and only if, languages K_1 and K_2 are nonconflicting, i.e., $\overline{K_1} \cap \overline{K_2} = \overline{K_1 \cap K_2}$.

III. System Models

A. Environment Automaton Model G_e

This paper deals with the mobile robot navigation problem in industrial environments where the structure rarely undergoes major modifications and, thus, it is reasonable to assume *a priori* knowledge of the environment and, also, that the visitable places do not change; for example, in a warehouse, the visitable places correspond to those places associated with all possible shelves the robot must access. We leverage this feature to model the environment by an automaton $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e}, X_{m_e})$, where the states in X_e are all possible robot poses (visitable places together with robot orientation in the navigation environment), Σ_e is the set of command events that correspond to those movements that connect the poses in X_e, the transition function f_e and the set of active events Γ_e are defined according to the environment connectivity. Finally, in order to compute a path to be followed by the mobile robot, x_{0_e} is defined as the robot pose at the beginning of the task and X_{m_e} is defined as the set of states that represent the complete execution of the task.

Notice that, although automaton G_e models the environment, its states represent the possible robot poses (positional coordinates and orientation) in the navigation environment, i.e., those poses the robot can visit when executes a string of command events formed from Σ_e^*. Notice that, since the robot must transport products, parts and raw materials around the plant, the positioning of the possible robot poses is dictated by both the environment structure and the places where machinery is laid in the plant. Automaton G_e can be constructed by using some roadmap construction technique, e.g., vertical cell decomposition [23], reduced visibility graphs [24], and generalized Voronoi diagrams [25], [26].

Let $x_1, x_{n+1} \in X_e$. A path in automaton G_e that takes the robot from state x_1 to state x_{n+1} has the form $x_1\sigma_1 x_2\sigma_2 x_3 \ldots \sigma_n x_{n+1}$, where, $\forall k \in \{1, \ldots, n\}$, $x_k \in X_e$, $\sigma_k \in \Sigma_e$, and $f_e(x_k, \sigma_k) = x_{k+1}$. Notice that there may exist several different paths that connect state x_1 to state x_{n+1}, and each of them is characterized by its corresponding string of command events $\sigma_1\sigma_2 \ldots \sigma_n \in \Sigma_e^*$. In order to compare different paths, we define the weight function

$$\begin{aligned} w\colon \Sigma_e &\to \mathbb{R} \\ \sigma &\mapsto w(\sigma) = c \end{aligned} \tag{1}$$

where $c \in \mathbb{R}_+$ represents the cost of executing the robot movement corresponding to command event σ. We, then, define the cost of executing a string $s = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma_e^*$ as follows:

$$J(s) = \sum_{i=1}^{n} w(\sigma_i). \tag{2}$$

B. Robot Model G_r

In order to construct a discrete event model for the robot, the features that are important for the correct planning and execution

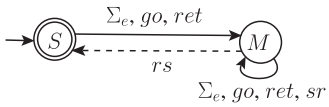

Fig. 1. Robot movement module G_{r_m}. Dashed lines represent transitions labeled with uncontrollable events.

TABLE I
ROBOT MOVEMENT MODULE EVENTS

Event	Description	Controllable
Σ_e	Set of environment automaton events	✓
sr	Stop the robot	✓
ret	Return to the last visited state	✓
go	Complete the last movement	✓
rs	Robot stopped	×

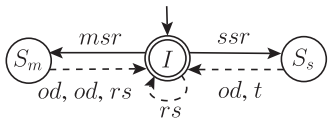

Fig. 2. Robot sensing module G_{r_s}. Dashed lines represent transitions labeled with uncontrollable events.

TABLE II
ROBOT SENSING MODULE EVENTS

Event	Description	Controllable
rs	Robot stopped	×
msr	Obstacle sensor information request while the robot is moving	✓
ssr	Obstacle sensor information request when the robot has stopped	✓
od	Obstacle detected	×
$\overline{od}$	No obstacle detected	×
t	Timeout	×

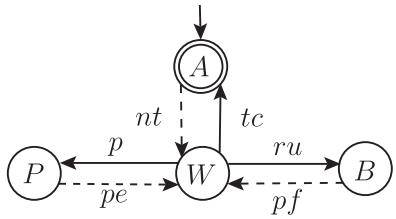

Fig. 3. Robot task manager module $G_{r_{\text{tm}}}$. Dashed lines represent transitions labeled with uncontrollable events.

of the navigation task are separately modeled by using automata. In this paper, we propose a robot automaton model, denoted by G_r, which is obtained by performing the following parallel composition:

$$G_r = G_{r_m} \| G_{r_s} \| G_{r_{\text{tm}}} \tag{3}$$

where G_{r_m}, G_{r_s}, and $G_{r_{\text{tm}}}$ model the robot movement, sensing, and task manager modules, respectively.

1) Robot Movement Module: Automaton G_{r_m}, depicted in Fig. 1, models the robot movement resources. The events of G_{r_m} are listed in Table I. In order to navigate in a given industrial environment, the robot must be able to execute the events in Σ_e, which is the set of events of automaton G_e that models the industrial environment. In addition, it also requires other command events to deal with unpredictable obstacles. When an obstacle is detected, command event sr is used to stop the robot in order to prevent a collision. Command events ret and go are used to return the robot to the last visited state and, to complete the movement interrupted by the obstacle, respectively. Automaton G_{r_m} has also the uncontrollable event rs, which is due to sensor readings, and represents the transition from state M, where the robot is moving, to state S, where the robot is stopped.

It is assumed that the robot has low level controllers that are able to execute the movement commands presented in Table I and, if the robot is performing some movement and receives a new movement command, it cancels the current movement command and executes the new one. Nevertheless, the events belonging to Σ_e remain controllable since the supervisors are able to prevent their occurrences.

2) Robot Sensing Module: Automaton G_{r_s}, depicted in Fig. 2, models the robot sensing resources. It is assumed that the robot has wheel encoders and at least one sensing system that is able to detect obstacles, such as sonars, laser rangefinders, or vision-based systems. The wheel encoders are used to determine when event rs occurs, that is, when the robot stops after finishing the last movement command. The obstacle detection sensors are used to monitor the presence of unpredictable obstacles that block the robot path by means of command events msr and ssr. Command event msr is used when the robot is moving to request the execution of a sensing routine, which, then, returns event od, $\overline{od}$, or rs (active event set of state S_m). Event od (resp $\overline{od}$) indicates that an obstacle is detected (resp. no obstacle detected), and event rs indicates that the robot stopped and, consequently, the sensor reading is no longer necessary. Command event ssr is used, when the robot is stopped, to start a sensing routine that keeps G_{r_s} in state S_s until either a timeout event t or no obstacle detection represented by event $\overline{od}$ occurs. The complete list of events of G_{r_2} is presented in Table II. Notice that events msr and ssr are controllable, whereas events rs, od, $\overline{od}$ and t are uncontrollable.

3) Robot Task Manager Module: Automaton $G_{r_{\text{tm}}}$, depicted in Fig. 3, models the robot resources associated with the management of the robot task. It has four states which represent the current robot status regarding task management, as follows:

1) robot available (A);
2) planning the robot trajectory (P);
3) executing the task (W); and
4) waiting for the removal of an obstacle (B).

The events of $G_{r_{\text{tm}}}$ are listed in Table III. Event nt is issued by an external agent requesting the execution of a new task by the robot. Command event p is used to start the path planning procedure. After that, the planner starts to execute the path planning and when it is completed, the planner sends to the robot a signal, which corresponds to the occurrence of event pe. Command event ru (request for unblocking the path) is a robot request to an external agent to remove the obstacle between

TABLE III
ROBOT TASK MANAGER MODULE EVENTS

Event	Description	Controllable
nt	New task received	×
tc	Robot reports task completion	✓
p	Execute the planning	✓
pe	Planning concluded	×
ru	Request for unblocking the last path	✓
pf	The last path is free	×

the current robot position and the last visited state. When the external agent removes the obstacle, it issues event pf, meaning that the path is free. Thus, events tc, p, and ru are controllable, and events nt, pe, and pf are uncontrollable.

IV. DES-Based Robot Navigation Architecture

A Robot navigation architecture is an structure composed by the modules that constitute a mobile robot navigation system (e.g., path planning, obstacle avoidance, etc.) and the framework used to combine them [7]. In this paper, we address the problem of the navigation of a mobile robot modeled by an automaton G_r, that navigates in an industrial environment modeled by an automaton G_e together with a cost function [function J, defined in (2)]. We assume that there may exist unpredictable permanent or intermittent obstacles in the environment, so that some transitions of the environment automaton are not allowed to fire either temporarily or definitely. We refer to such transitions as blocked transitions.

The following robot tasks are considered.

1) Task 1. This task is completed when the robot reaches some state (pose) belonging to a set $X_{\text{goal}} \subseteq X_e$.
2) Task 2. This task is completed after the robot visits all states (pose) belonging to a set $X_{\text{goal}} \subseteq X_e$.

Since the robot is required to visit only one state in X_{goal} when it executes Task 1, it is necessary to determine the path starting at the current state of the robot and ending at one of the states in X_{goal} that minimizes the cost function J. On the other hand, when the robot executes Task 2, it needs to visit all states in X_{goal}, regardless of the ordering, by following a path that minimizes the cost function J.

A. Navigation Architecture

The navigation architecture proposed in this paper is formed by a planner and a modular supervisory control structure, as shown in Fig. 4. An advantage of this structure is that it allows a decentralized implementation, where the modular supervisor is embedded in the mobile robot, whereas the planner runs in an external agent, which suits very well to the design problem addressed here suitable, since it makes easier to adapt the proposed navigation architecture to other environments.

The navigation process starts when the robot is available and an external agent assigns a new task to the robot. This assignment is modeled by the occurrence of event nt of automaton $G_{r_{\text{mt}}}$ of Fig. 3. Then, robot G_r generates event p (execute planning), which carries the following information.

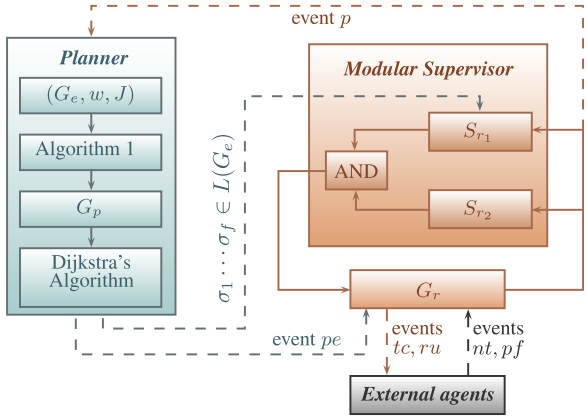

Fig. 4. Proposed navigation architecture.

1) The robot current state.
2) The last task assigned to the robot and its respective set of target states X_{goal}.
3) Set T_b, which is formed with those transitions identified as blocked, being initialized as an empty set and modified by the robot when event t occurs in G_{r_s}, i.e., the blocked transition of G_e is added to T_b when the robot detects a permanent obstacle. The blocked transition is determined from the last state of G_e visited by the robot and the command event in Σ_e whose execution was interrupted by the obstacle detection. It is worth remarking that we can limit the time interval in which a transition stays in T_b with a view to checking if this transition is still blocked during the execution of a future robot task.

According to the diagram depicted in Fig. 4, when the planner receives event p, it runs a computer application to determine the path to be followed by the robot. Notice that the planner is composed by the following.

1) Environment automaton G_e.
2) Weight function w and cost function J, defined in accordance with (1) and (2), respectively.

After the planning is finished, the planner sends event pe to automaton G_r to inform that the planning has been concluded. The planner also sends the string of command events that corresponds to the computed path to be used to design supervisor S_{r_1}; thus, a new supervisor S_{r_1} is computed after each new path planning completion.

The modular supervisory control $S_{r_1} \wedge S_{r_2}$ ensures the correct navigation of the robot in the presence of unpredictable permanent or intermittent obstacles; S_{r_1} acts so as to enforce the robot to follow the path computed by the planner, whereas S_{r_2} ensures that design specifications SP_1–SP_7 (to be presented in Section IV-C), are achieved.

B. Path Planning Procedure

The first step in the path planning procedure is to compute, using Algorithm 1, a refined automaton G_p so that the language marked by G_p is formed by those strings of command events that can be executed by the robot with a view to completing the robot task. Algorithm 1 starts by setting up as initial and marked

Algorithm 1: Computation of automaton G_p.

Inputs:
- $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e}, X_{m_e})$: environment automaton
- The robot current pose
- X_{goal}: the set of target states
- task_type $\in \{$Task 1, Task 2$\}$;
- T_b: set of blocked transitions of G_e.

Output: Automaton $G_p = (X_p, \Sigma_p, f_p, \Gamma_p, x_{0_p}, X_{m_p})$.

begin
- Set the robot current pose and X_{goal} as the initial and marked states of G_e, respectively;
- $G_p \leftarrow G_e$;
- **if** $T_b \neq \emptyset$ **then**
  - **for** *every transition* $x \xrightarrow{\sigma} y \in T_b$ **do**
    - $f_p(x, \sigma) \leftarrow$ undefined;
    - $\Gamma_p(x) \leftarrow \Gamma_p(x) \setminus \{\sigma\}$;
- $G_p \leftarrow CoAc[Ac(G_p)]$;
- **if** *task_type* $=$ *Task 1* **then**
  - **if** *the set of states of* G_p *is empty* **then**
    - **return** "Error: impossible task"
  - **else**
    - **return** G_p;
- **else if** *task_type* $=$ *Task 2* **then**
  - **if** *the set of states of* G_p *does not contain* X_{goal} **then**
    - **return** "Error: impossible task"
  - **else**
    - $G_{temp} \leftarrow G_p$;
    - **for** *every state* $x \in X_{goal}$ **do**
      - $G_x = (X_x, \Sigma_x, f_x, \Gamma_x, x_{0_x}, X_{m_x}) \leftarrow G_p$;
      - **for** *every event* $\sigma \in \Sigma_e$ **do**
        - $f_x(x, \sigma) \leftarrow x$;
      - $\Gamma_x(x) \leftarrow \Sigma_e$;
      - Redefine the set of marked states of G_x as $\{x\}$;
      - $G_{temp} \leftarrow G_{temp} \| G_x$
      - **if** *the set of marked states of* G_{temp} *is empty* **then**
        - **return** "Error: impossible task"
    - $G_p \leftarrow CoAc(G_{temp})$;
    - **return** G_p

states of G_e the robot current pose and X_{goal}, respectively, and, after that, G_e is assigned to G_p. If $T_b \neq \emptyset$, then we remove from G_p the transitions in T_b, and set G_p as CoAc[Ac(G_p)]. When Task 1 is assigned, automaton G_p must be nonempty. In this case, $L_m(G_p)$ will be formed by strings of command events that can be used to complete Task 1. When Task 2 is assigned, automaton G_p must be modified to form a new automaton whose marked language is formed by those strings in $L_m(G_p)$ that correspond to paths containing all of the states in X_{goal}. This can be done as follows. Let us assume that $X_{\text{goal}} = \{x_1, \ldots, x_n\}$. Then, for each $x_i \in X_{\text{goal}}$, automaton G_{x_i}, whose marked language contains all strings of $L_m(G_p)$ that correspond to paths that visit state x_i, is constructed. Subsequently, automaton G_p is redefined as $G_p \leftarrow G_p \| G_{x_1} \| \ldots \| G_{x_n}$. As a consequence, the language marked by the new automaton G_p will be formed by those strings that correspond to the paths that contain all of the states in X_{goal}, since it is equal to the intersection of the languages marked by the initial G_p and by automata G_{x_i}, for every $x_i \in X_{\text{goal}}$.

Finally, by applying Dijkstra's algorithm [27] using, as input, G_p and its initial state, we determine the marked state of G_p that is nearest the initial state, and the string $\sigma_1 \ldots \sigma_f \in \Sigma_e^*$

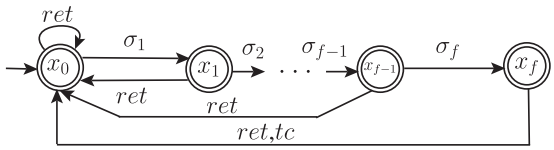

Fig. 5. Automaton $H_{\text{spec},1}$ used to synthesize supervisor S_{r_1}.

that corresponds to the feasible path that minimizes the cost function J, being, therefore, the solution to the path planning for the robot navigation problem.

C. Design of Modular Supervisor $S_{r_1} \wedge S_{r_2}$

In order to design supervisors S_{r_1} and S_{r_2}, we initially construct simple automata that capture the essence of the specifications we want to ensure by using these supervisors. We, then, combine these automata with G_r using the parallel composition to obtain the system desired behavior.

Let us first consider the design of S_{r_1}. According to the diagram of Fig. 4, we intend to synthesize a supervisor S_{r_1} that enforces the robot to follow the path computed by the planner. Let $s = \sigma_1 \ldots \sigma_i \sigma_{i+1} \ldots \sigma_f \in L(G_e)$ be the string of command events computed by the planner. Notice that $\sigma_i \in \Sigma_e \subset \Sigma_r$, for $i = 1, \ldots, f$, and thus, the behavior of G_r must be restricted to ensure that sequence s is executed. This can be done by creating the specification automaton $H_{\text{spec},1}$ depicted in Fig. 5, which is formally defined as $H_{\text{spec},1} = (X_1, \Sigma_1, f_1, \Gamma_1, x_0, X_1)$, where $X_1 = \{x_0, x_1, \ldots, x_f\}$, $\Sigma_1 = \{ret, tc\} \cup \Sigma_e$, and f_1 is defined, as follows:

$$f_1(x_i, \sigma) = \begin{cases} x_{i+1}, & \text{if} \sigma = \sigma_{i+1} \\ x_0, & \text{if} (\sigma = ret) \vee ((\sigma = tc) \wedge (x_i = x_f)) \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Notice that events ret and tc have been included in $H_{\text{spec},1}$ in order to account for possible obstacle detection, which makes the robot abort the execution of the planned trajectory, and to report that the task has been completed, respectively.

Automaton H_1 that marks the applicable language requirement K_1 is computed by performing the parallel composition between automaton G_r, obtained (3), and $H_{\text{spec},1}$, as follows:

$$H_1 = G_r \| H_{\text{spec},1}.$$

It is worth remarking that the set of events of automaton G_r is $\Sigma_r = \Sigma_e \cup \{sr, ret, go, rs, msr, ssr, od, \overline{od}, t, nt, tc, p, pe, ru, pf\}$. Thus, if $P_1 : \Sigma_r^* \rightarrow \Sigma_1^*$, we can state that

$$K_1 = P_1^{-1}[L_m(H_{\text{spec},1})] \cap L_m(G_r). \tag{4}$$

Notice that, to achieve the requirement imposed by language K_1, only events in Σ_1 may be disabled. Since all events in Σ_1 are controllable, it is not difficult to conclude that K_1 is controllable. In addition, because all states of $H_{\text{spec},1}$ are marked, K_1 is, by construction, $L_m(G_r)$-closed. Then, an automaton realization of a nonblocking supervisor S_{r_1} such that $L_m(S_{r_1}/G_r) = K_1$ can be obtained from $H_{\text{spec},1}$ by adding self-loops labeled by the events in $\Sigma_r \setminus \Sigma_1$ to all of its states.

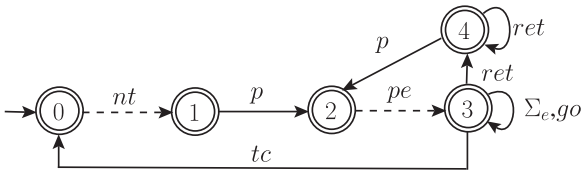

Fig. 6. Automaton $H_{\text{spec},2}$ used to synthesize supervisor S_{r_2}. Dashed lines represent transitions labeled with uncontrollable events.

Let us now consider the synthesis of supervisor S_{r_2}, which deals, among other requirements, with permanent and intermittent obstacles. In practice, the robot classifies a previously detected obstacle as permanent or intermittent by using the sensing routine started by command event ssr, that is, when this sensing routine returns timeout event t, the obstacle is said to be permanent, and, when it returns event $\overline{od}$, the obstacle is said to be intermittent. In order to achieve the desired behavior, the following specifications are enforced.

1) SP_1. The robot can perform the command events in $\Sigma_e \cup \{ret, go\}$ only after it receives a new task and the trajectory has already been planned.
2) SP_2. After the execution of command event ret, which, according to specification SP_6, will only be executed after a permanent obstacle is detected, the robot cannot perform movement commands in $\Sigma_e \cup \{go\}$ before a new trajectory is computed by the planner.
3) SP_3. In order to prevent the wrong functioning of low level controllers that execute the robot movements, a command event in $\Sigma_e \cup \{ret, go\}$ cannot be sent before the execution of the previous movement has been either completed or aborted by command sr (stop the robot).
4) SP_4. In order to prevent collisions, the robot continuously checks the existence of obstacles in the trajectory, and, when an obstacle is detected, it must stop.
5) SP_5. After the robot stops due to an obstacle detection, it must distinguish between intermittent and permanent obstacles in order to avoid unnecessary computations of new trajectories. In addition, if the obstacle is intermittent, the robot must try to complete the interrupted movement when the obstacle is no longer detected.
6) SP_6. When the robot detects a permanent obstacle, it must return to the last visited state in G_e, which corresponds to the last pose visited in the environment, by using command event ret.
7) SP_7. When the robot detects a permanent obstacle while it executes the movements associated with command event ret, it must request an external agent to remove this obstacle by means of event ru.

We will now construct specification automata that capture the essence of specifications SP_i, $i = 1, \ldots, 7$. We first construct automaton $H_{\text{spec},2}$ depicted in Fig. 6, that accounts for specifications SP_1 and SP_2 whose set of events is $\Sigma_2 = \Sigma_e \cup \{nt, p, pe, tc, ret, go\}$. From Fig. 6, we can see that: 1) events in $\Sigma_e \cup \{ret, go\}$ can only be executed at state 3 of $H_{\text{spec},2}$, which is reached only after the occurrence of pe (planning executed), and; 2) after the occurrence of event ret, all events in $\Sigma_e \cup \{go\}$ remain disabled until the conclusion of a new path planning. We added a self-loop labeled by event ret at state 4, because it may be necessary to perform several occurrences of this event until the robot reaches the last visited state after the detection of a permanent obstacle. This is so because a new obstacle can be detected while the robot is returning to the last visited state. This issue is addressed in the next specification automaton.

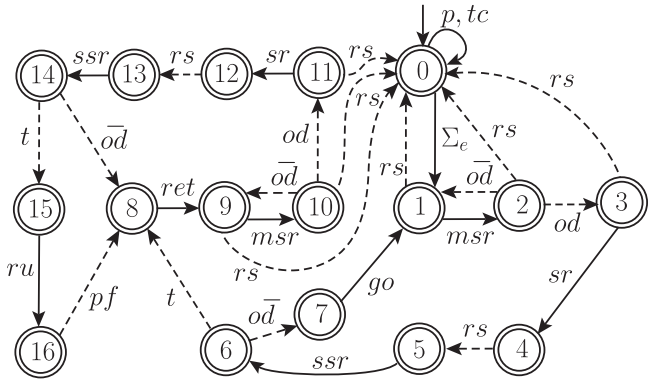

Fig. 7. Automaton $H_{\text{spec},3}$ used to synthesize supervisor S_{r_2}. Dashed lines represent transitions labeled with uncontrollable events.

Automaton $H_{\text{spec},3}$, depicted in Fig. 7, accounts for specifications SP_3—SP_7. Its set of events is $\Sigma_3 = \Sigma_r \setminus \{nt, pe\}$. The states of $H_{\text{spec},3}$ correspond to the following situations.

1) State 0 represents the case when the robot has stopped without detecting obstacles.
2) States 1 and 2 correspond to the case when the robot is executing the movements associated with the command events in Σ_e.
3) States 3 to 16 are associated with the procedure to handle obstacles.

After the robot executes an event in Σ_e, leading to state 1 of $H_{\text{spec},3}$, it must request obstacle sensor information by means of event msr, therefore, moving to state 2. If no obstacle is detected, event $\overline{od}$ occurs; this procedure is, then, repeated until the completion of the current movement, which is indicated by the occurrence of event rs, therefore, leading to state 0. If an obstacle is detected, $H_{\text{spec},3}$ evolves to state 3 through the transition labeled by event od. The transition from state 3 to 0 labeled by event rs models the case when, after an obstacle detection, the robot stops before the command event sr is issued. In this case, we assume that the detected obstacle did not prevent the execution of the previous movement, and thus, it can be disregarded. On the other hand, the transition from state 3 to 4 labeled by event sr represents the control action that enforces the robot to stop, whose completion is confirmed by the occurrence of event rs (robot stopped) at state 4. At state 5 of $H_{\text{spec},3}$, the robot requests obstacle sensor information by means of event ssr in order to determine whether the detected obstacle is permanent or intermittent. Then, $H_{\text{spec},3}$ remains in state 6 until the occurrence of either $\overline{od}$ or t. The occurrence of $\overline{od}$ means that the previously detected obstacle is intermittent and no longer blocks the path; thus, the robot can complete the

movement stopped due to the obstacle, which is done by means of command event go. On the other hand, if the timeout event t occurs, it can be inferred that the previously detected obstacle is permanent, and, so, the robot needs to return to the last visited state in G_e in order to compute a new trajectory, which is done by executing event ret. If the robot detects a permanent obstacle while it is returning to the last visited state, that is, if event t occurs at state 15 of $H_{\text{spec},3}$, then, it executes event ru by sending a request to unblock the path, and remains at state 16 until the occurrence of event pf, which means that the path has been unblocked. After the occurrence of event pf, the robot executes event ret again to return to the last visited state of G_e.

Automaton H_2 that marks the applicable language requirement K_2 is computed by performing the parallel composition between G_r, $H_{\text{spec},2}$, and $H_{\text{spec},3}$, as follows:

$$H_2 = G_r \| H_{\text{spec},2} \| H_{\text{spec},3}.$$

If $P_2 : \Sigma_r^* \to \Sigma_2^*$ and $P_3 : \Sigma_r^* \to \Sigma_3^*$ denote projections, then

$$K_2 = L_m(G_r) \cap P_2^{-1}[L_m(H_{\text{spec},2})] \cap P_3^{-1}[L_m(H_{\text{spec},3})]. \quad (5)$$

It can be check that K_2 is controllable and $L_m(G_r)$-closed.

An automaton realization of a nonblocking supervisor S_{r_2} such that $L_m(S_{r_2}/G_r) = K_2$ can be obtained by calculating the parallel composition $H_{\text{spec},2} \| H_{\text{spec},3}$. Notice that the set of events of $H_{\text{spec},2} \| H_{\text{spec},3}$ is equal to Σ_r, and, thus, we do not need to add self-loops to it in order to obtain a realization of supervisor S_{r_2}.

According to modular supervisory control theory presented in Section II-B, the modular supervisory architecture constructed by the conjunction of S_{r_1} and S_{r_2} is such that $L_m(S_{r_1} \wedge S_{r_2}/G_r) = K_1 \cap K_2$, where K_1 and K_2 are defined in (4) and (5), respectively. In addition, it can be verified that the admissible languages K_1 and K_2 are nonconflicting, which ensures that the conjunctive modular supervisor $S_{r_1} \wedge S_{r_2}$ is nonblocking, i.e., $L(S_{r_1} \wedge S_{r_2}/G) = \overline{K_1 \cap K_2}$.

Fig. 8 shows automaton $S_{r_1} \wedge S_{r_2}/G_r$ that models the closed-loop behavior in the case that the robot trajectory computed by the planner is equal to $\sigma_1\sigma_2 \ldots \sigma_f \in L(G_e)$. Such an automaton can be obtained by performing the following parallel composition:

$$S_{r_1} \wedge S_{r_2}/G_r = G_r \| H_{\text{spec},1} \| H_{\text{spec},2} \| H_{\text{spec},3}. \quad (6)$$

Notice that, although the automaton shown in Fig. 8 grows linearly with the length of the robot trajectory (the colored part), the parallel computation presented in (6) is not required to implement the modular architecture proposed here, since the designs of S_{r_1} and S_{r_2} are based on different automaton specifications, $H_{\text{spec},1}$ and $H_{\text{spec},2} \| H_{\text{spec},3}$, respectively.

V. Performance Analysis of the Proposed Algorithm

We present, in this section, a performance analysis of the algorithm proposed here. We first present the space complexity analysis and, in the sequel, a time complexity analysis.

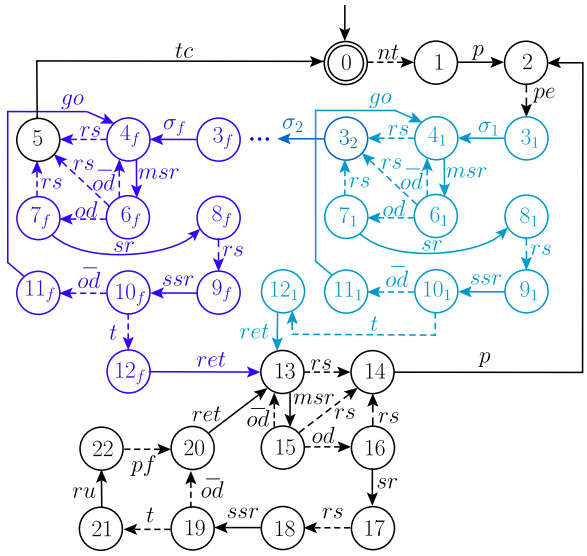

Fig. 8. Closed-loop behavior for robot trajectory $\sigma_1\sigma_2 \ldots \sigma_f \in L(G_e)$. Dashed lines represent transition labeled with uncontrollable events.

A. Scalability Analysis

In the DES-based robot navigation architecture proposed here, the number of transitions of the environment automaton model G_e is $O(|X_e|\,|\Sigma_e|)$, where $|X_e|$ and $|\Sigma_e|$ are the number of the robot poses of interest and command events that are necessary to the robot navigation, respectively. On the other hand, the robot model G_r, obtained in accordance with (3), has 24 states and $132 + |\Sigma_e|$ transitions, and, thus, the robot model is $O(|\Sigma_e|)$.

The computational complexity of the planning procedure presented in Section IV-B depends on the type of the robot task. When the robot performs Task 1, Algorithm 1 is executed in time $O(|T_b| + |X_e|\,|\Sigma_e| + |X_{\text{goal}}|)$ and generates an automaton G_p whose number of states $|X_p|$ is $O(|X_e|)$. Assuming that the priority queue used in Dijkstra's algorithm is implemented as a Fibonacci heap, the shortest path in automaton G_p is found in time $O(|X_p| \log |X_p| + |X_p|\,|\Sigma_e|)$. Therefore, the path planning procedure in the case of Task 1 is executed in time $O(|T_b| + |X_e|(|\Sigma_e| + \log |X_e|) + |X_{\text{goal}}|)$. When the robot performs Task 2, Algorithm 1 is executed in time $O(|T_b| + 2^{|X_{\text{goal}}|}\,|X_e|\,|\Sigma_e|)$ and the number of states of automaton G_p, $|X_p|$, is $O(2^{|X_{\text{goal}}|}\,|X_e|)$. As a consequence, in the case of Task 2, the path planning procedure is $O(|T_b| + 2^{|X_{\text{goal}}|}\,|X_e|(|\Sigma_e| + \log |X_e| + |X_{\text{goal}}|))$.

Regarding the complexity of the design of modular supervisor $S_{r_1} \wedge S_{r_2}$, it can be seen that the number of states and transitions of automaton $H_{\text{spec},1}$ are both $O(\|\text{path}\|)$, where $\|\text{path}\|$ denotes the number of command events whose robot must execute to complete the task and was obtained in the path planning procedure. As a consequence, supervisor S_{r_1} is $O(\|\text{path}\|)$. In addition, it can be seen that automaton $H_{\text{spec},2} \| H_{\text{spec},3}$ has 22 states and $32 + |\Sigma_e|$ transitions and, consequently, supervisor S_{r_2} is $O(|\Sigma_e|)$.

Based on the computational complexities presented above, it can be concluded that the proposed DES-based navigation

TABLE IV
SUMMARY OF THE RESULTS FOR TIME COMPLEXITY ANALYSIS

Group	G_e	Parameters				Planning
	$\lvert X_e\rvert/\lvert T_e\rvert$	Task	x_{0_e}	$\lvert X_{\text{goal}}\rvert$	$\lvert\text{obst}\rvert$	(s)
1	273/1000	1	1	1	0	0.024
	273/1000	1	M_1	1	0	0.016
	273/1000	1	29	1	0	0.025
	273/1000	1	M_6	1	0	0.016
	273/1000	1	M_4	1	0	0.016
	273/1000	1	M_2	1	0	0.016
2	39/140	1	1	1	0	0.003
	78/284	1	1	1	0	0.007
	117/428	1	1	1	0	0.008
	156/57	1	1	1	0	0.017
	195/714	1	1	1	0	0.018
	234/857	1	1	1	0	0.020
	273/1000	1	1	1	0	0.023
3	39/140	2	1	2	0	0.364
	78/284	2	1	2	0	0.424
	117/428	2	1	2	0	1.085
	156/57	2	1	2	0	1.401
	195/714	2	1	2	0	1.922
	234/857	2	1	2	0	2.001
	273/1000	2	1	2	0	2.564
4	273/1000	1	1	2	0	0.022
	273/1000	1	1	3	0	0.024
	273/1000	1	1	4	0	0.024
	273/1000	1	1	5	0	0.023
	273/1000	1	1	6	0	0.024
	273/1000	1	1	7	0	0.024
5	39/140	2	1	2	0	0.364
	39/140	2	1	3	0	0.572
	39/140	2	1	4	0	1.148
	39/140	2	1	5	0	2.024
	39/140	2	1	6	0	5.440
	39/140	2	1	7	0	19.473

architecture scales well with respect to the size of the environment since the aforementioned computational complexities increase with a factor less than $|X_e|^2$ and $|\Sigma_e|$ when either $|X_e|$ or $|\Sigma_e|$ increases, and $|X_e|(|X_e| + |\Sigma_e|)$, when both $|X_e|$ and $|\Sigma_e|$ increase. On the other hand, when the robot performs Task 2, the approach may not scale well with respect to the number of target states since the computational effort to compute the robot path increases exponentially with the number of target states ($|X_{\text{goal}}|$).

B. Time Complexity Analysis

The results obtained from a series of numerical experiments carried out on a laptop with an Intel Core i5-4210U Processor, 8 Gb DDR3 RAM, are shown in Table IV. It is divided in five groups, as follows: Group 1, that consists in running Task 1 cases in an environment modeled by an automaton G_e with 273 states and 1000 transitions, all having the same target state but different initial robot positions; Groups 2 and 3 that show the influence of changes in the environment size in the execution times of Tasks 1 and 2, respectively; and Groups 4 and 5, that show the performance results for Tasks 1 and 2, respectively, as the number of target states in X_{goal} increases. Notice that the times taken to perform the path planning in Group 1 are approximately the same for all simulations of Group 1. This result has already been expected since, as shown in Section V-A,

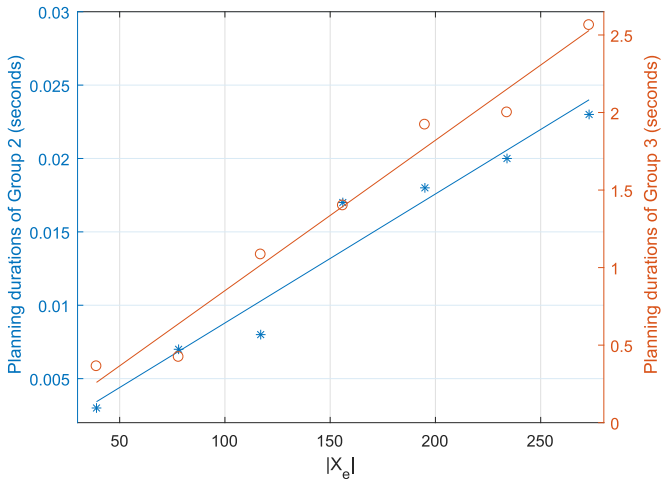

Fig. 9. Times taken to perform the path planning versus the size of the environment state space for Groups 2 and 3.

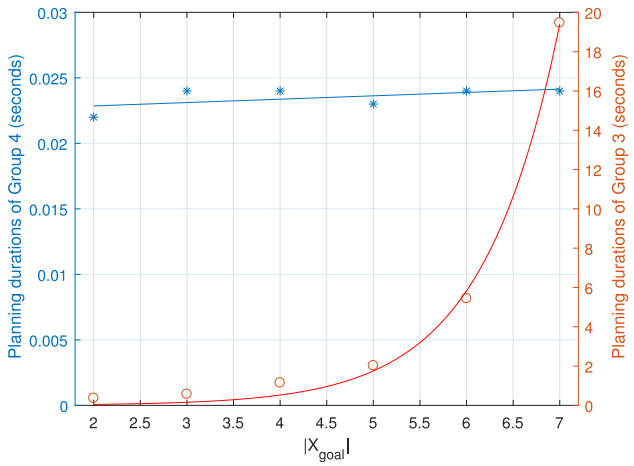

Fig. 10. Times taken to perform the path planning versus the number of target states for Groups 4 and 5.

the computational effort of the path planning procedure for cases of Task 1 is predominantly determined by input parameters $|X_e|$, $|\Sigma_e|$, $|T_b|$, and $|X_{\text{goal}}|$. As far as Groups 2 and 3 are concerned, the time taken to perform the path planning grows almost linearly with the increase in the environment state space, as we can see from the plots depicted in Fig. 9. Regarding Groups 4 and 5, as shown in Fig. 10, the time for path planning is approximately the same in all cases of Group 4 whereas the state size of G_p grows exponentially with the number of target states in tasks of type 2. Notice that these results are in accordance with the theoretical ones of Section V-A since automaton G_p is as large as G_e in type 1 tasks but grows exponentially with the number of target states for type 2 tasks.

VI. SIMULATION RESULTS

Consider the hypothetical industrial environment depicted in Fig. 11, which resembles a smart factory composed by a conveyor belt ①, which is a loading and unloading terminal, shelves ⑤ and ⑥ that are used to store raw materials and parts processed either by the computer numerically controlled milling

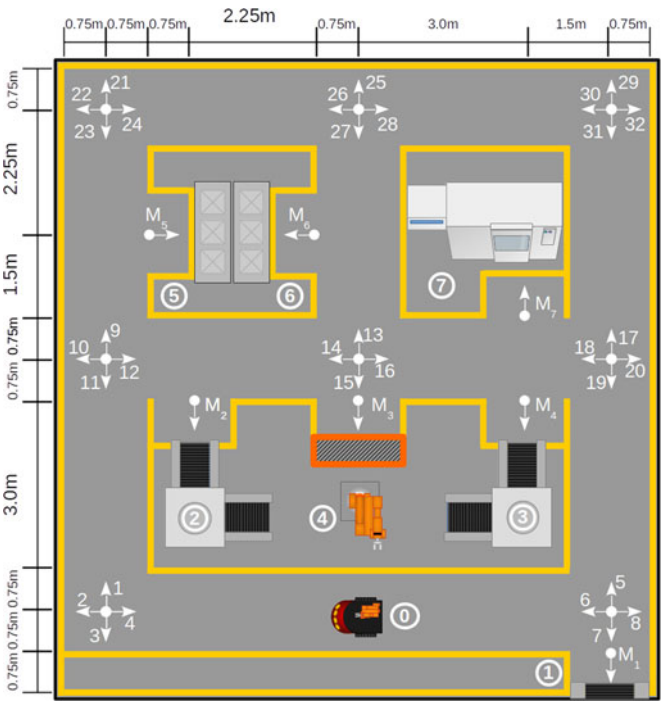

Fig. 11. Map of the environment. The arrows represent the possible robot poses (states of automaton G_e): the tail indicates the positional coordinate and the direction corresponds to the robot pose.

machine ⑦ or the painting machines ② and ③. The robot arm ④ is used either to transfer parts from one paint machine to the other, or to reject those painted parts that do not meet some quality standard to the orange rectangle. Notice that mobile robot ⓪ must be at pose M_i to interact with element ⓘ, $i = 1, \ldots, 7$.

The automaton that models this environment is $G_e = (X_e, \Sigma_e, \Gamma_e, x_{0_e}, X_{m_e})$, where

1) $X_e = X_{\text{int}} \dot{\cup} X_{\text{ps}}$ with $X_{\text{int}} = \{1, 2, 3, \ldots, 32\}$ and $X_{\text{ps}} = \{M_1, M_2, M_3, \ldots, M_7\}$ are formed with those poses that correspond to the corridor intersections and those poses that allow the robot to pick up and store parts, respectively;
2) Σ_e, formed by the events listed in Table V;
3) Γ_e: $X_e \rightarrow 2^{\Sigma_e}$, presented in Table VI for all $x \in X_e$;
4) f_e, defined, for each state $x \in X_e$, according to the active events presented in Table VI—for example, for $x = 1$, transition function f_e is defined for events $m4.5$, $t180$, $t90$, and $t90^-$, and, as it can be seen, with the help of Fig. 11, $f_e(1, m4.5) = 9$, since command event $m4.5$ models a 4.5 m forward movement, $f_e(1, t180) = 3$, since command event $t180$ corresponds to a 180° rotational movement, and $f_e(1, t90) = 2$ (resp. $f_e(1, t90^-) = 4$) since command event $t90$ (resp. $t90^-$) is associated with a 90° counterclockwise (resp. clockwise) rotational movement; and
5) x_{0_e} and X_{m_e}, respectively, which are defined by the planner at each task assigned to the robot, as described in Section IV-B.

Notice that events smi, $i = 1, 2, \ldots, 13$ are composed by a sequence of translational and rotational movements, where the translational movements do not necessarily correspond to some event md.

TABLE V
ENVIRONMENT AUTOMATON EVENTS Σ_e

Event	Description	$w(.)$
$m0.75$	Move robot 0.75 m	0.76
$m4.5$	Move robot 4.5 m	4.51
$m9.0$	Move robot 9.0 m	9.01
$t90$	Turn robot 90° (counterclockwise)	0.46
$t90^-$	Turn robot −90° (clockwise)	0.46
$t180$	Turn robot 180°	0.91
$sm1$	Turn robot 180° and move 0.75 m	1.67
$sm2$	Move 1.5 m, turn 90°, and move 0.75 m	2.73
$sm3$	Move 1.5 m, turn −90°, and move 0.75 m	2.73
$sm4$	Move 2.25 m, turn 90°, and move 0.75 m	3.48
$sm5$	Move 2.25 m, turn −90°, and move 0.75 m	3.48
$sm6$	Move 3.00 m, turn 90°, and move 0.75 m	4.23
$sm7$	Move 3.00 m, turn −90°, and move 0.75 m	4.23
$sm8$	Turn 180°, move 0.75 m, turn 90°, and move 1.5 m	3.64
$sm9$	Turn 180°, move 0.75 m, turn −90°, and move 1.5 m	3.64
$sm10$	Turn 180°, move 0.75 m, turn 90°, and move 2.25 m	4.39
$sm11$	Turn 180°, move 0.75 m, turn −90°, and move 2.25 m	4.39
$sm12$	Turn 180°, move 0.75 m, turn 90°, and move 3.00 m	5.14
$sm13$	Turn 180°, move 0.75 m, turn −90°, and move 3.00 m	5.14

TABLE VI
ACTIVE EVENTS OF THE STATES OF G_e

X_e	Γ_e	X_e	Γ_e
1	$\{t90, t90^-, t180, m4.5\}$	21	$\{t90, t90^-, t180\}$
2	$\{t90, t90^-, t180\}$	22	$\{t90, t90^-, t180\}$
3	$\{t90, t90^-, t180\}$	23	$\{t90, t90^-, t180, m4.5, sm4\}$
4	$\{t90, t90^-, t180, m9.0\}$		
5	$\{t90, t90^-, t180, m4.5\}$	24	$\{t90, t90^-, t180, m4.5\}$
6	$\{t90, t90^-, t180, m9.0\}$	25	$\{t90, t90^-, t180\}$
7	$\{t90, t90^-, t180, m0.75\}$	26	$\{t90, t90^-, t180, m4.5\}$
8	$\{t90, t90^-, t180\}$	27	$\{t90, t90^-, t180, m4.5, sm5\}$
9	$\{t90, t90^-, t180, m4.5, sm5\}$		
10	$\{t90, t90^-, t180\}$	28	$\{t90, t90^-, t180, m4.5\}$
11	$\{t90, t90^-, t180, m4.5\}$	29	$\{t90, t90^-, t180\}$
12	$\{t90, t90^-, t180, m4.5, sm3\}$	30	$\{t90, t90^-, t180, m4.5\}$
13	$\{t90, t90^-, t180, m4.5, sm4\}$	31	$\{t90, t90^-, t180, m4.5\}$
14	$\{t90, t90^-, t180, m4.5, sm6\}$	32	$\{t90, t90^-, t180\}$
15	$\{t90, t90^-, t180, m0.75\}$	M_1	$\{sm1\}$
16	$\{t90, t90^-, t180, m4.5, sm6, sm7\}$	M_2	$\{sm8, sm11\}$
		M_3	$\{sm1\}$
17	$\{t90, t90^-, t180, m4.5\}$	M_4	$\{sm9, sm12\}$
18	$\{t90, t90^-, t180, m4.5, sm2, sm3\}$	M_5	$\{sm10, sm11\}$
		M_6	$\{sm10, sm11\}$
19	$\{t90, t90^-, t180, m4.5\}$	M_7	$\{sm8, sm13\}$
20	$\{t90, t90^-, t180\}$		

The values of the weight function w are listed in the third column of Table V, being defined as follows:

1) $w(md) = d + 0.01$, where $d \in \{0.75, 4.5, 9.0\}$ is the distance to be traveled;
2) $w(t\theta) = |\theta|/200 + 0.01$, where $\theta \in \{90, 180\}$ is the rotation angle; and
3) $w(smk) = \sum_i (|\theta_i|/200 + 0.01) + \sum_j (d_j + 0.01)$, $k = 1, 2, \ldots, 13$, where θ_i (resp. d_j) are all rotational (resp. translational) movements present in smk.

In order to illustrate the results of the paper, we will perform two pairs of simulation, corresponding to two different tasks assigned to the robot, assuming, initially, no obstacle and, then,

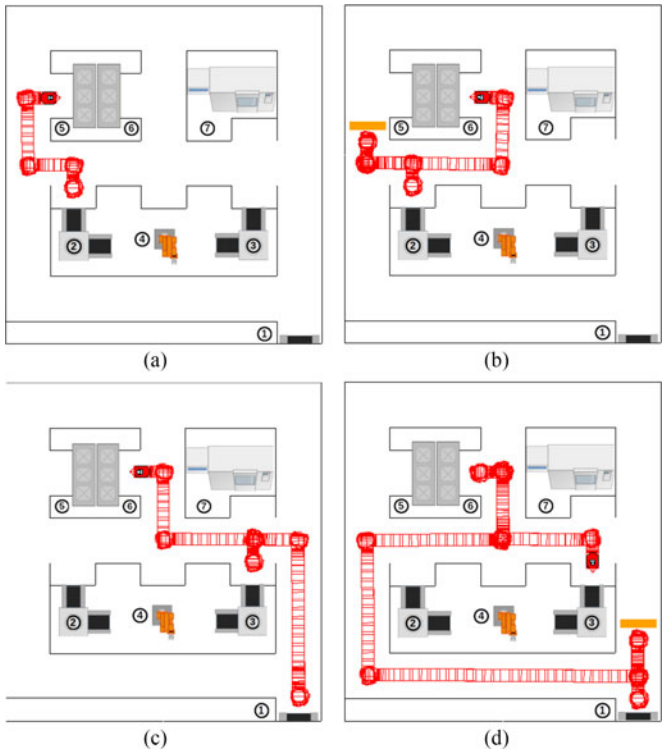

Fig. 12. Simulation results: (a) Type 1 task from pose M_2 to $X_{\text{goal}} = \{M_5, M_6\}$ without obstacle and (b) with and (c) Type 2 task from pose M_1 to $X_{\text{goal}} = \{M_3, M_6\}$ without obstacle and (d) with obstacle.

with some obstacle in the robot trajectory. The simulations were performed using MobileSim 0.7.5 software for a virtual Pioneer P3DX mobile robot.

For the first simulation, we assume that the robot is initially at pose M_2 and has just picked up a processed part in painting machine ② when it receives a request to store the part in shelf ② or ②, which is the closest. Since this is a Type 1 task, the mobile robot must determine which shelf is the nearest, compute the shortest path to there, and perform the computed string of command movements. According to Fig. 8, after event nt is issued, the robot sends event p to the planner, which starts the computation of the optimal path taking into account automaton G_e, the set of target states $X_{\text{goal}} = \{M_5, M_6\}$, and the set of blocked transitions $T_b = \emptyset$ (no obstacle is initially assumed to exist). As a consequence, automaton G_p, computed by applying Algorithm 1, is equal to G_e with initial and marked states defined as M_2 and $\{M_5, M_6\}$, respectively. After automaton G_p is computed, Dijkstra's algorithm is applied to find the path from the initial state to one of the marked states of G_p that minimizes cost function J given in (2), yielding string $s_1 = sm8\, t90^{-} sm5$. Once the planning has been concluded, event pe is issued, and so, the robot executes string s_1 under the control action of supervisor $S_{r_1} \wedge S_{r_2}$, designed according to Section IV-C, performing the path shown in Fig. 12(a).

For the second simulation, we assume that the robot must execute the same task as before but now we added an obstacle between the poses of states 9 and M_5, as highlighted in orange in Fig. 12(b). Since, initially, the robot does not know the obstacle, the string of command events obtained by the path planning procedure is equal to string $s_1 = sm8\, t90^{-} sm5$ computed in the first simulation. However, while the robot is executing the movements associated with command $sm5$, it executes command event msr and, thus, its sonar detects the obstacle that is blocking the path, which makes event od occur. As a consequence, the robot stops immediately after that (event sr). Thus, since the obstacle is permanent, the robot waits until the occurrence of timeout t (defined, empirically, as 5s), and, in the sequel, executes event ret to reach state 9, which is the last visited state before command event $sm5$ has been executed. The planner then computes a new path adding the blocked transitions to T_b, i.e., $T_b = \{(9, sm5, M_5)\}$, where (x, σ, y) denotes a transition of G_e from state x to state y labeled by σ. The new string computed by the planner is equal to $s_2 = t90^{-} m4.5\, t90\, sm4$, which corresponds to the path depicted in Fig. 12(b). Since this new trajectory is free of obstacles, the robot is then able to arrive at state M_6.

For the third and fourth simulations, we assume that the robot is initially at pose M_1 when it receives a request to pick up some amount of paint at the conveyor belt ① and to deliver part of this paint to machine ③ and to store the remainder in shelf ⑥, which corresponds to a Type 2 task. When there is no obstacle in the environment, the robot performs the path depicted in Fig. 12(c) by executing the string of command events $s_3 = t180\, m4.5\, t90\, sm2\, sm12\, t90^{-} sm4$ obtained by the planner for $T_b = \emptyset$, $x_{0_e} = M_1$, and $X_{m_e} = \{M_3, M_6\}$. When an obstacle is added to the environment between the poses of states 5 and 17, the robot starts executing s_3, as before, because it is not, at first, aware of the existence of the obstacle, but detects the obstacle while executing the movement corresponding to event $m4.5$. As a consequence, the robot returns to pose 5 and new command string $s_4 = t90\, m9.0\, t90^{-} m4.5\, t90^{-} m4.5\, t90\, sm4\, sm11\, t90\, sm13$ is computed by the planner for $T_b = \{(5, m4.5, 17)\}$, $x_{0_e} = 5$, and $X_{m_e} = \{M_3, M_6\}$, which allows the robot to successfully complete the task, as shown in Fig. 12(d).

VII. Concluding Remarks

We addressed, in this paper, a general methodology for mobile robot navigation in industrial environments in which the free behavior of the robot and the specifications are all automaton based. The theory of supervisory control of discrete event systems was used to obtain a modular supervisory controller that ensures the correct navigation of the robot in the presence of unpredictable obstacles. The proposed approach provides a general modeling framework allowing the implementation of specifications by means of modules that depend solely on the type of task the robot will perform and on the environment. One future research direction would be the application of the DES-based architecture proposed here in cooperative control of multirobot systems.

Acknowledgment

The authors would like to thank anonymous reviewers for their important suggestions which considerably improved the quality of the paper.

References

[1] A. Gilchrist, *Industry 4.0: The Industrial Internet of Things*, 1st ed. Berkeley, CA, USA: Apress, 2016.

[2] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, "Towards smart factory for industry 4.0: A self-organized multi-agent system with big data based feedback and coordination," *Comput. Netw.*, vol. 101, pp. 158–168, 2016.

[3] V. Jirkovsky, M. Obitko, and V. Marik, "Understanding data heterogeneity in the context of cyber-physical systems integration," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 660–667, Apr. 2017.

[4] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[5] J. Wan *et al.*, "A manufacturing big data solution for active preventive maintenance," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 2039–2047, Aug. 2017.

[6] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[7] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Bradford Book, 2004.

[8] D. Nakhaeinia, S. Tang, S. M. Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots," *Int. J. Phys. Sci.*, vol. 6, no. 2, pp. 169–174, 2011.

[9] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Math. Control, Signals, Syst.*, vol. 1, no. 1, pp. 13–30, 1988.

[10] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY, USA: Springer, 2008.

[11] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 40, no. 9, pp. 1555–1575, Sep. 1995.

[12] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[13] M. V. Moreira and J. C. Basilio, "Bridging the gap between design and implementation of discrete-event controllers," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 48–65, Jan. 2014.

[14] I. Antunes, L. K. Carvalho, and J. C. Basilio, "A stochastic Petri net model for simulation-based performance analysis of public bicycle sharing systems," in *Proc. Int. Conf. Autom. Sci. Eng.*, Fort Worth, TX, USA, 2016, pp. 433–439.

[15] B. Liu, M. Ghazel, and A. Toguyéni, "Model-based diagnosis of multi-track level crossing plants," *IEEE Trans. Intell. Trans. Syst.*, vol. 17, no. 2, pp. 546–556, Feb. 2016.

[16] E. Roszkowska, "Supervisory control for multiple mobile robots in 2d space," in *Proc. Int. Workshop Robot Motion Control*, Bukowy Dworek, Poland, 2002, pp. 187–192.

[17] E. Fabre and L. Jezequel, "Distributed optimal planning: An approach by weighted automata calculus," in *Proc. 48th IEEE Conf. Decision Control Held Jointly with the 28th Chin. Control Conf.*, Shanghai, China, 2009, pp. 211–216.

[18] M. Kloetzer and C. Mahulea, "Multi-robot path planning for syntactically co-safe ltl specifications," in *Proc. Int. Workshop Discr. Event Syst.*, Xi'an, China, 2016, pp. 452–458.

[19] J. Iqbal, S. Khan, N. Zafar, and F. Ahmad, "Modeling supervisory control of autonomous mobile robots using graph theory, automata and z notation," *J. Amer. Sci.*, vol. 8, no. 12, pp. 799–804, 2012.

[20] J. Košecká and R. Bajcsy, "Discrete event systems for autonomous mobile agents," *Robot. Auton. Syst.*, vol. 12, pp. 187–198, 1994.

[21] J. Goryca and R. Hill, "Formal synthesis of supervisory control software for multiple robot systems," in *Proc. Amer. Control Conf.*, Washington, DC, USA, 2013, pp. 125–131.

[22] R. Hill and S. Lafortune, "Scaling the formal synthesis of supervisory control software for multiple robot systems," in *Proc. Amer. Control Conf.*, Seattle, WA, USA, 2017, pp. 3840–3847.

[23] B. Chazelle, "Approximation and decomposition of shapes," in *Algorithmic and Geometric Aspects of Robotics*, J. T. Schwartz and C. K. Yap, Eds. Hillsdale, NJ, USA: Lawrence Erlbaum Assoc., 1987, pp. 145–185.

[24] S. K. Ghosh and D. M. Mount, "An output sensitive algorithm for computing visibility graphs," *SIAM J. Comput.*, vol. 20, pp. 888–910, 1991.

[25] C. O'Dunlaing and C. K. Yap, "A retraction method for planning the motion of a disc," *J. Algorithms*, vol. 6, pp. 104–111, 1982.

[26] M. Sharir, "Algorithmic motion planning," in *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. Boca Raton, FL, USA: CRC Press, Inc., 1997, pp. 733–754.

[27] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.

Antonio G. C. Gonzalez was born on March 1, 1994 in Fortaleza, Brazil. He received the Control and Automation Engineer degree from the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, in 2017. He is currently working toward the M.Sc. degree in control at the Federal University of Rio de Janeiro.

His main interests include supervisory control of discrete-event systems and mobile robotics.

Marcos V. S. Alves was born on March 5, 1988 in Aracaju, Brazil. He received the electronic engineering degree from the Federal University of Sergipe, Sergipe, Brazil, in 2011, and the M.Sc. and D.Sc. degrees in control from the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, in 2014 and 2017, respectively.

His main interests include supervisory control and failure diagnosis of discrete event systems and cyber-physical system security.

Gustavo S. Viana was born on July 9, 1990 in Rio de Janeiro, Brazil. He received the electrical engineering degree and the M.Sc. degree in control from the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, in 2012 and 2014, respectively. He is currently working toward the D.Sc. degree in control at the same university.

His main interests include failure diagnosis of discrete event systems, max-plus algebra, and development of teaching techniques in engineering education.

Lilian K. Carvalho (M'07) was born on March 11, 1979 in São Paulo, Brazil. She received the electronic engineering degree, the M.Sc. degree and the D.Sc. degree in control from the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, in 2003, 2005, and 2011, respectively.

Since 2011, she has been an Associate Professor with the Department of Electrical Engineering, Federal University of Rio de Janeiro. From September, 2014 to December 2015, she spent a sabbatical year with the University of Michigan, Ann Arbor, MI, USA. Her main interests include fault diagnosis of discrete-event systems, cyber-attacks, and the development of control laboratory techniques.

Dr. Carvalho is the IEEE Rio de Janeiro Section WIE Chair.

João C. Basilio (M'13–SM'16) was born on March 15, 1962, in Juiz de Fora, Brazil. He received the electrical engineering degree from the Federal University of Juiz de Fora, Juiz de Fora, Brazil, in 1986, the M.Sc. degree in control from the Military Institute of Engineering, Rio de Janeiro, Brazil, in 1989, and the Ph.D. degree in control from Oxford University, Oxford, U.K., in 1995.

He began his career in 1990 as an Assistant Lecturer with the Department of Electrical Engineering, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, where he is currently a Full Professor in Control. Since February, 2014, he has been the Dean of Polytechnic School of UFRJ. From September, 2007 to December, 2008, he spent a sabbatical leave at the University of Michigan, Ann Arbor, MI, USA, and was an Invited Professor of École Centrale de Lille, University of Lille, France, during September 2016. His current interests include fault diagnosis and supervisory control of discrete-event systems.

Prof. Basilio is the recipient of the Correia Lima Medal.