Control Engineering Practice 81 (2018) 97–104

Contents lists available at ScienceDirect

Control Engineering Practice

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

Application of online supervisory control of discrete-event systems to multi-robot warehouse automation

Yuta Tatsumoto [a,*], Masahiro Shiraishi [a], Kai Cai [a], Zhiyun Lin [b]

[a] Department of Electrical and Information Engineering, Osaka City University, Japan
[b] Hangzhou Dianzi University, China

ARTICLE INFO

Keywords:
Supervisory control
Discrete-event systems
Warehouse automation
Online control

ABSTRACT

In this paper we present an online supervisory control approach, based on limited lookahead policy, that is amenable for the control of multi-agent discrete-event systems. We then apply this online control scheme to model and control a warehouse automation system served by multiple mobile robots; the effectiveness of this scheme is demonstrated through a case study. Moreover, we build an experiment testbed for testing the validity of our proposed method with implementation on real robots.

1. Introduction

Supervisory control theory of discrete-event systems (DES) was first proposed by Ramadge and Wonham in the 1980s (Ramadge & Wonham, 1987), with the aim to formalizing general (high-level) control principles for a wide range of application domains. In this theory, DES are modeled as finite-state automata, and their behaviors represented by regular languages. The control feature is that certain events (or state transitions) can be disabled by an external supervisor to enforce a desired behavior. This feature leads to the fundamental concept of *language controllability*, which determines the existence of a supervisor that suitably disables a series of events in order to satisfy an imposed control specification. For a comprehensive account of supervisory control theory, the reader is referred to Wonham and Cai (2018); also see Wonham, Cai, and Rudie (2018) for a recent historical overview of the theory.

While supervisory control of DES is theoretically sound, it is often computationally infeasible for practical systems that are large and complex, due to the notorious problem of state explosion (Gohari & Wonham, 2000). Indeed, the computational complexity of synthesizing a supervisor is exponential in the number of plant components. In addition, systems that are time-varying or subject to unknown changes are also unsuitable to be dealt with by supervisory control theory, because a supervisor is synthesized offline and consequently cannot account for changes in operation.

To address large and time-varying DES, online supervisory control based on *limited lookahead policy* was proposed in Chung, Lafortune, and Lin (1992, 1993, 1994). This control scheme generates at the current state a limited-step-ahead projection of the plant's behavior, and determines based on the projected behavior the next control action to satisfy an imposed specification. A new projection is generated after each occurrence of event, so time-varying changes in the system can be taken into account in this scheme. In particular, Chung et al. (1993, 1994) presented recursive strategies to make the online supervisory synthesis more efficient. Other extensions of the online supervisory control can be found in Boroomand and Hashtrudi-Zad (2013), Hadj-Alouane, Lafortune, and Lin (1996), Kumar, Cheung, and Marcus (1998) and Winacott and Rudie (2009). In Hadj-Alouane et al. (1996), online supervisory control of partially observed DES was addressed where some occurrences of events are unobservable. In Kumar et al. (1998), a variant method was reported for generating a limited-step-ahead projection of the plant's behavior. In Winacott and Rudie (2009), online control of probabilistic DES was investigated where event occurrences are random accordingly to given probabilities. Finally in Boroomand and Hashtrudi-Zad (2013), online control was extended to deal with robust supervisory control where the plant model is uncertain. Common in all the online supervisory control methods above (Boroomand & Hashtrudi-Zad, 2013; Chung et al., 1992, 1993, 1994; Hadj-Alouane et al., 1996; Kumar et al., 1998; Winacott & Rudie, 2009), the plant model is assumed to be given or already computed.

This paper adapts the online supervisory control approach to the case of multi-agent DES, i.e. plant consisting of multiple components, and applies the approach to warehouse automation systems served by multiple mobile robots. In recent years warehouse automation has received significant interest from both academia and industries, due to

* Corresponding author.
E-mail address: tatsumoto@c.info.eng.osaka-cu.ac.jp (Y. Tatsumoto).

https://doi.org/10.1016/j.conengprac.2018.09.003
Received 19 March 2018; Received in revised form 3 July 2018; Accepted 5 September 2018

its central role in the rapidly growing e-commerce, supply chain, and material handling enterprise. A landmark of such examples is Kiva systems (Wurman, D'Andrea, & Mountz, 2008), which dispatches hundreds of robots to serve the logistics in Amazon's distribution centers.

The contributions of this paper are threefold. First, we present a key modification to the existing online supervisory control approach such that the approach is amenable to multi-agent DES. The approach in Boroomand and Hashtrudi-Zad (2013), Chung et al. (1992, 1993, 1994) , Hadj-Alouane et al. (1996), Kumar et al. (1998) and Winacott and Rudie (2009) is not suitable for multi-agent DES because it assumes that the plant model is already given or can be computed, and then generates a limited-step-ahead projection of the plant model. For multi-agent DES, however, the plant model is the synchronous product of the component agents, which itself may not be feasibly computable particularly when the number of agents is large. To circumvent this problem, we propose to first generate limited-step-ahead projections of each agent model, and then compute the synchronous product of the projected agent models. This is computationally more efficient than the existing approaches in Boroomand and Hashtrudi-Zad (2013), Chung et al. (1992, 1993, 1994), Hadj-Alouane et al. (1996), Kumar et al. (1998) and Winacott and Rudie (2009). Second, we show how a warehouse automation system served by multiple robots can be modeled as a multi-agent DES, and demonstrate through a case study the effectiveness of applying the adapted online supervisory control approach to control the automation system. Finally, we build a testbed, consisting of multiple LEGO MINDSTORMS EV3 robots (https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313), that serves to experimentally test the validity of our proposed online supervisory control approach for multi-agent DES.

We note that Goryca and Hill (2013), Hill and Lafortune (2016, 2017), Lopes, Trenkwalder, Leal, Dodd, and Groß(2016), Su (2013), Su and Lennartson (2017), Su and Lin (2013) and Ware and Su (2016) also studied supervisory control for multi-agent DES. In Ware and Su (2016), the authors presented a method that assigns distinct priorities to all agents and then incrementally synthesizes supervisors for solving a scheduling problem. This method is efficient, but need not find a supervisor for scheduling even when a schedule exists. In Lopes et al. (2016), the authors applied supervisory control methods to achieving various cooperative behaviors of swarm robots (including segregation, aggregation, object clustering, and formations), and demonstrated their results on real multi-robot systems. In Goryca and Hill (2013) and Hill and Lafortune (2016, 2017), the authors proposed an abstraction-based method to efficiently synthesize supervisors for multi-agent DES, and implemented the method into a software whose efficiency was tested through a multi-robot planning case study. Finally in Su (2013), Su and Lennartson (2017) and Su and Lin (2013), the authors proposed a scalable control design for a type of multi-agent DES, where an "agent" was not just a plant component, but indeed a plant of its own including an imposed specification. The "agents" were instantiated from a template; for the template, under certain conditions, an algorithm was proposed to design a supervisor whose instantiation was shown to work for each "agent". In all the references above, the proposed control synthesis methods are offline; although these methods addressed in one way or another the issue of computational efficiency, they cannot handle multi-agent DES that are time-varying and subject to unknown changes. By contrast, our proposed method for multi-agent DES is online, and therefore not only reduces computational effort, but also deals with dynamic changes in the system.

The rest of this paper is organized as follows. In Section 2 we introduce the adapted online supervisory control of multi-agent DES. In Section 3 we model a warehouse automation system by multi-agent DES, and apply online supervisory control for its control. Moreover, we present a testbed for experimental validation of online supervisory control for warehouse automation. Finally in Section 4 we state our conclusions.

2. Online supervisory control of multi-agent DES

In standard supervisory control (Ramadge & Wonham, 1987; Wonham and Cai, 2018), the plant to be controlled is modeled by a finite state *automaton*

$$\mathbf{G} := (Q, \Sigma, \delta, q_0, Q_m)$$

where Q is the finite state set, $q_0 \in Q$ the *initial state*, $Q_m \subseteq Q$ the set of *marker states*, Σ the finite *event set*, and $\delta : Q \times \Sigma \to Q$ the (partial) *state transition function*. Letting Σ^* denote the set of all finite-length strings of events in Σ, we extend δ such that $\delta : Q \times \Sigma^* \to Q$ and write $\delta(q, s)!$ to mean that $\delta(q, s)$ is defined.

The closed behavior of $\mathbf{G}$ is the set of all strings that can be generated by $\mathbf{G}$:

$$L(\mathbf{G}) := \{s \in \Sigma^* | \delta(q_0, s)!\}.$$

On the other hand, the *marked behavior* of $\mathbf{G}$ is the subset of strings that can reach a marker state:

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) | \delta(q_0, s) \in Q_m\} \subseteq L(\mathbf{G}).$$

$\mathbf{G}$ is *nonblocking* if $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$ ($\overline{\cdot}$ means *prefix closure*), namely every string in the closed behavior may be completed to a string in the marked behavior.

The event set Σ of $\mathbf{G}$ is partitioned into a subset Σ_c of *controllable* events and a subset Σ_u of *uncontrollable* events. A language $E \subseteq \Sigma^*$ is said to be *controllable* (with respect to $\mathbf{G}$) if $\overline{E}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{E}$, i.e.

$$(\forall s \in \Sigma^*, \forall \sigma \in \Sigma) s \in \overline{E}, \sigma \in \Sigma_u, s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{E}.$$

Let $K \subseteq L_m(\mathbf{G})$ be a specification language imposed on the plant $\mathbf{G}$. Denote by $C(K)$ the family of controllable sublanguages of K, i.e.

$$C(K) := \{K' \subseteq K \mid \overline{K'}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K'}\}.$$

Then the supremal controllable sublanguage of K exists and is given by $\sup C(K) = \cup\{K' \mid K' \in C(K)\}$. Let **SUP** be a (nonblocking) automaton such that $L_m(\mathbf{SUP}) = \sup C(K)$. We call **SUP** the supervisor for plant $\mathbf{G}$ that enforces $\sup C(K) \subseteq L_m(\mathbf{G})$. The control action of **SUP** after an arbitrary string $s \in L(\mathbf{G})$ is to enable the events in

$$\gamma := \{\sigma \in \Sigma_u \mid s\sigma \in L(\mathbf{G})\} \cup \{\sigma \in \Sigma_c \mid s\sigma \in L(\mathbf{SUP})\}.$$

To introduce online supervisory control based on limited lookahead policy, we need an operation that 'truncate' an automaton from a specified state to limited step ahead. Given an automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, a state $q \in Q$, and the step number $N \geq 1$, the 'truncation' operation $f^N(\mathbf{G}, q)$ on $\mathbf{G}$ at state q defines a new automaton

$$f^N(\mathbf{G}, q) = \mathbf{G}^N(q) := (Q^N, \Sigma^N, \delta^N, q_0^N, Q_m^N)$$

where

$$\begin{aligned} Q^N &= \{q' \in Q \mid (\exists s \in \Sigma^*) q' = \delta(q, s) \ \&\ |s| \leq N\} \\ \Sigma^N &= \{\sigma \in \Sigma \mid (\exists q' \in Q^N) \delta(q', \sigma)! \ \&\ \delta(q', \sigma) \in Q^N\} \\ \delta^N &= \{(q_1, \sigma, q_2) \mid q_1, q_2 \in Q^N \ \&\ \sigma \in \Sigma^N \ \&\ \delta(q_1, \sigma) = q_2\} \\ q_0^N &= q \\ Q_m^N &= Q_m \cap Q^N. \end{aligned}$$

In the above definition of Q^N, $|s|$ denotes the *length* of string s. Note also that the marker state set Q_m^N may be empty.

In a multi-agent DES, the plant consists of $n(> 1)$ component agents, where each agent $k(\in \{1, \ldots, n\})$ is modeled by a finite state automaton $\mathbf{G}_k = (Q_k, \Sigma_k, \delta_k, q_{k,0}, Q_{k,m})$. The plant model $\mathbf{G}$ is the *synchronous product* (Wonham and Cai, 2018) of the n component agents, written $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m) = \mathbf{G}_1 \| \cdots \| \mathbf{G}_n$. The event set Σ_k of each agent $\mathbf{G}_k$ is partitioned into a controllable subset and an uncontrollable subset, i.e. $\Sigma_k = \Sigma_{k,c} \cup \Sigma_{k,u}$; hence $\Sigma_c = \cup_{k\in\{1,\ldots,n\}} \Sigma_{k,c}$, $\Sigma_u = \cup_{k\in\{1,\ldots,n\}} \Sigma_{k,u}$, and $\Sigma = \cup_{k\in\{1,\ldots,n\}} \Sigma_k$.

When the number n of component agents is large, their synchronous product, namely the computation of the overall plant model $\mathbf{G}$, is computationally expensive, if not impossible. As a consequence, the online supervisory control in Chung et al. (1992, 1993, 1994) that assumes $\mathbf{G}$ is available is not directly suitable for multi-agent DES. By contrast, we propose the following modified and more efficient online supervisory control, which first truncates each agent's automaton into limited step ahead and then form the synchronous product of the truncated automata.

Multi-agent online supervisory control procedure:

Let $s \in L(\mathbf{G})$ and plant $\mathbf{G}$ is at state $q := \delta(q_0, s)$. Since $\mathbf{G}$ is the synchronous product of $\mathbf{G}_1, \dots, \mathbf{G}_n$, state q is the n-tuple $q = (q_1, \dots, q_n)$, where q_k is the current state of component $\mathbf{G}_k$ ($k \in \{1, \dots, n\}$). Let $N(\geq 1)$ be the size of the lookahead window.

(1) For each component agent $\mathbf{G}_k$ ($k \in \{1, \dots, n\}$), generate the N-step-ahead behavior from the current state q_k by applying the truncation operation

$$\mathbf{G}_k^N(q_k) = f^N(\mathbf{G}_k, q_k).$$

(2) Compute the plant model $\mathbf{G}^N(q)$ as the synchronous product of all truncated component models $\mathbf{G}_k^N(q_k)$:

$$\mathbf{G}^N(q) = \mathbf{G}_1^N(q_1) \| \cdots \| \mathbf{G}_n^N(q_n).$$

(3) The states that are exactly N-step-ahead from q in $\mathbf{G}^N(q)$ are called *pending states*. A pending state may transition to an undesirable state by an uncontrollable event, or it may not be able to reach a marker state. There are two opposite attitudes as to how to deal with pending states, for which we follow Chung et al. (1992).

(a) Conservative attitude: all pending states are deemed undesirable state and are removed, i.e.

$$\mathbf{G}(q) = f^{N-1}(\mathbf{G}^N(q), q) = (Q^N, \Sigma^N, \delta^N, q, Q_m^N).$$

(b) Optimistic attitude: all pending states are deemed desirable states and are treated as marker states, i.e.

$$\mathbf{G}'(q) = f^N(\mathbf{G}^N(q), q) = (Q^N, \Sigma^N, \delta^N, q, Q_m'^N);$$
$$\mathbf{G}(q) = (Q^N, \Sigma^N, \delta^N, q, Q_m^N)$$

where

$$Q_m^N := Q_m'^N \cup \{q' \in Q^N \mid (\exists s \in \Sigma^*) q' = \delta^N(q, s) \ \&\ |s| = N\}.$$

(4) Let $K \subseteq L_m(\mathbf{G})$ be a specification, and $\mathbf{K} = (P, \Sigma, \pi, p_0, P_m)$ be an automaton such that $L_m(\mathbf{K}) = K$. Since $s \in L_m(\mathbf{G})$ has occurred in $\mathbf{G}$, the current state of $\mathbf{K}$ is $p := \pi(p_0, s)$. The truncated specification for $\mathbf{G}(q)$ is determined again according to two opposite attitudes.

(a) Conservative attitude:

$$\mathbf{K}(p) = f^{N-1}(\mathbf{K}^N(p), p) = (P^N, \Sigma^N, \pi^N, p, P_m^N).$$

(b) Optimistic attitude:

$$\mathbf{K}'(p) = f^N(\mathbf{K}^N(p), p) = (P^N, \Sigma^N, \pi^N, p, P_m'^N);$$
$$\mathbf{K}(p) = (P^N, \Sigma^N, \pi^N, p, P_m^N)$$

where

$$P_m^N := P_m'^N \cup \{p' \in P^N \mid (\exists s \in \Sigma^*) p' = \pi^N(p, s) \ \&\ |s| = N\}.$$

(5) Compute the supremal controllable sublanguage $\sup C(L_m(\mathbf{K}(p)))$ with respect to $\mathbf{G}(q)$, and let $\mathbf{SUP}(q) = (X^N, \Sigma^N, \xi^N, q, X_m^N)$ be the supervisor for $\mathbf{G}(q)$ that enforces $\sup C(L_m(\mathbf{K}(p)))$, i.e. $L_m(\mathbf{SUP}(q)) = \sup C(L_m(\mathbf{K}(p)))$. The control action of $\mathbf{SUP}(q)$ at state q is to enable the events in

$$\gamma(q) := \{\sigma \in \Sigma_c \mid \xi^N(q, \sigma) \in L(\mathbf{SUP}(q))\}$$
$$\cup \{\sigma \in \Sigma_u \mid \delta^N(q, \sigma) \in L(\mathbf{G}(q))\}.$$

One event $\sigma \in \gamma(q)$ will be executed such that the plant $\mathbf{G}$ transitions to the next state $\delta(q, \sigma)$. Correspondingly, component agents $\mathbf{G}_k$ will transition to the next state $\delta_k(q_k, \sigma)$ (provided $\delta_k(q_k, \sigma)$!). The above procedure repeats with string $s\sigma$.

The above online supervisory control scheme is the original one in Chung et al. (1992) tailored specifically to multi-agent DES. In particular, this scheme first generates a truncated automaton of each component agent and then form the plant model by synchronous product of the truncated automata; this is more efficient than computing the plant model as synchronous product of the non-truncated automata of the component agents. In addition, the implementation of the above online supervisory control scheme may be made more efficient by adopting the recursive computation or variable limited lookahead steps as in Chung et al. (1993, 1994). On the other hand, this online scheme has the same weakness as Chung et al. (1992): nonblocking controlled behavior need not be ensured, and to ensure nonblocking sufficiently long lookahead step may be required (see also Boroomand & Hashtrudi-Zad, 2013). This is an important issue that we aim to address in our immediate future work, by adapting the results in Boroomand and Hashtrudi-Zad (2013), Chung et al. (1992) to our setup.

Remark 1. We analyze the computational complexity of the proposed multi-agent online supervisory control procedure. Let $n(q)$ denote the state number of $\mathbf{G}(q)$ computed in Step 3), and $m(p)$ the state number of $\mathbf{K}(p)$ in Step 4). Then according to Ramadge and Wonham (1987), the computation of $\mathbf{SUP}(q)$ in Step 5) has complexity $O(n(q) \cdot m(p))$. In typical cases, the specification $\mathbf{K}(p)$ can be chosen as a subautomaton of $\mathbf{G}(q)$ (i.e. $m(p) \leq n(q)$), and hence the complexity of Step 5) is simply $O(n(q)^2)$. To generate $\mathbf{G}(q)$ in Step 3), however, we need to first compute $\mathbf{G}^N(q)$ in Step 2). Let $l(q)$ be the maximum state number of $\mathbf{G}_i^N(q_i)$, $i = 1, \dots, n$. Then the computation of $\mathbf{G}^N(q)$ has complexity $O(l(q)^n)$ (Gohari & Wonham, 2000). Therefore, the overall complexity of the multi-agent online supervisory control procedure is $O(\max_q l(q)^{2n})$.

The only difference in the original online supervisory control algorithm in Chung et al. (1992) is that of computing $\mathbf{G}^N(q)$. Specifically, $\mathbf{G}^N(q)$ in Chung et al. (1992) was computed as the N-step-ahead truncation of $\mathbf{G} = \mathbf{G}_1 \| \cdots \| \mathbf{G}_n$, i.e. $\mathbf{G}^N(q) := f^N(\mathbf{G}, q)$. Let k be the maximum state number of $\mathbf{G}_i$, $i = 1, \dots, n$. Then the computation of $\mathbf{G}$, and thus of $\mathbf{G}^N(q)$, has complexity $O(k^n)$ (Gohari & Wonham, 2000). Overall, the complexity of the online supervisory control algorithm in Chung et al. (1992) (indeed in all Boroomand & Hashtrudi-Zad, 2013; Chung et al., 1992, 1993, 1994; Hadj-Alouane et al., 1996; Kumar et al., 1998; Winacott & Rudie, 2009) is $O(k^{2n})$.

In comparison, although the original online algorithm and our proposed one both have complexity exponential in the number of agents, since typically $\max_q l(q) \ll k$ our proposed algorithm effectively reduces computational effort.

3. Warehouse automation application

In today's unprecedentedly growing e-commerce, supply chain, and material handling industries, warehouse automation is key to achieving significant efficiency in logistics. A landmark example is Amazon's Kiva systems (Wurman et al., 2008). In this section we show how to apply the multi-agent online supervisory control scheme to model and control a warehouse served by multiple mobile robots.

3.1. Warehouse configuration

Different warehouses have different configurations. To make our presentation concrete, we adopt the grid-type layout as displayed in Fig. 1. Mobile robots are assumed to be waiting for tasks at the top area, items to be picked up stored in the black-rectangle areas, and item-delivery destination (where a human worker is operating) at the bottom.

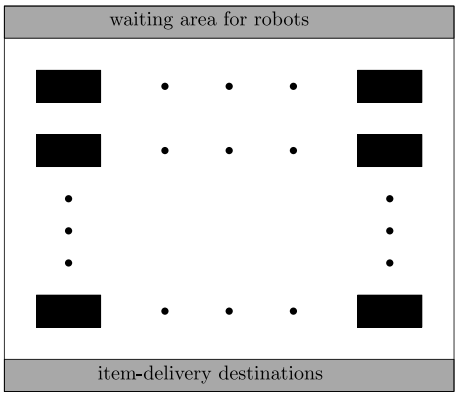

Fig. 1. Warehouse configuration: items to be picked up are stored in black-rectangle areas.

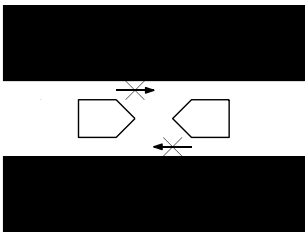

Fig. 2. Deadlock (or blocking): neither robots can move forward and no robot can finish its delivery.

When a robot is assigned a task, it is given 3 locations:

(i) start location: a location just below the top waiting area;

(ii) item location: a location in one of the black-rectangle areas;

(iii) goal location: a location just above the bottom item-delivery destination.

Upon being assigned a task, a robot should make its way first to the item location, picking up the item, and then to the goal location.

During navigation in the warehouse, a robot must avoid all item-storage (black-rectangle) areas at all times, only except when it needs to enter such an area to pick up its assigned item. We assume for simplicity that a robot enters an item-storage area only from above (i.e. a robot only moves downward to enter a black-rectangle area). It is also assumed that a robot can move forward, turn left, turn right, but never move backward.

Despite that the configuration described above is to some degree *ad hoc*, we can study several general control issues when the warehouse is served concurrently by multiple robots for its item-pickup/delivery logistics. These issues are:

(i) safety: robots must not collide with one another;

(ii) deadlock-free: robots must not block each other (in aisles between item-storage areas as shown in Fig. 2) such that no one can accomplish its delivery;

(iii) efficiency: the total time of finishing item delivery of all robots should be as short as possible.

In the sequel we shall apply the multi-agent online supervisory control scheme to address the above issues. Specifically, the safety requirement will be imposed explicitly as a *mutual exclusion* specification that prohibits using the same location by more than one robot at any time. Deadlock-free will be enforced by requiring that the synthesized controlled behavior be nonblocking. Finally efficiency, while difficult to be dealt with by the untimed supervisory control, will be reflected in modeling the individual robots.

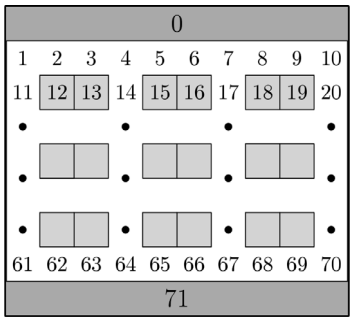

Fig. 3. Warehouse grid assigned with numbers.

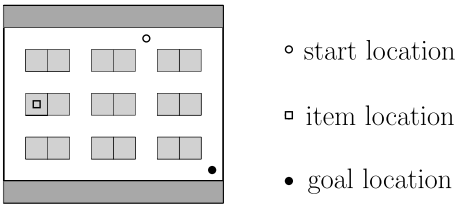

Fig. 4. Example: start, item, and goal locations assigned to a particular robot.

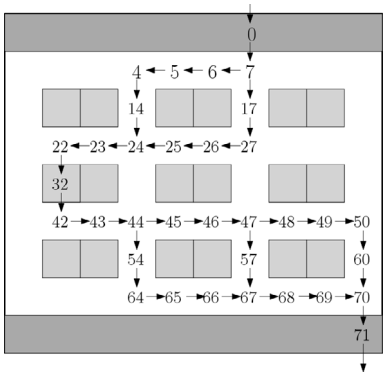

Fig. 5. Automaton model of the robot in Fig. 4.

Table 1
Event numbers of each robot $k \in \{1, \ldots, n\}$.

go north	$k \times 10 + 1$
go east	$k \times 10 + 3$
go south	$k \times 10 + 5$
go west	$k \times 10 + 7$

3.2. Automata models of robots

Consider there are $n(> 1)$ robots serving the warehouse. To propose a model for each robot, we consider a further concretized layout displayed in Fig. 3. Assign natural numbers to each cell of the grid; these will be used as state numbers. The waiting area for robots at the top is assigned "0", which is the initial state for all robots. On the other hand, the item delivery destination at the bottom, assigned in this case "71", is the only marker state for each robot.

In the grid a robot may move north, south, west, or east. For robot $k(\in \{1, \ldots, n\})$, designate the events with numbers as shown in Table 1. As is reasonable for this application, all events are assumed to be controllable.

When a task is assigned to robot k (i.e. a start location $q_{k,start}$, an item location $q_{k,item}$, and a goal location $q_{k,goal}$ are given), calculate the following *shortest paths* (those with the least number of events):

(i) find *all* shortest paths between $q_{k,start}$ and the location just *above* $q_{k,item}$ (say $q^{\uparrow}_{k,item}$);

(ii) find *all* shortest paths between the location just *below* $q_{k,item}$ (say $q^{\downarrow}_{k,item}$) and $q_{k,goal}$.

These shortest paths reflect the efficiency issue mentioned in the preceding subsection.[1] As an example, in Fig. 4 a robot is assigned a start location "7", item location "32", and goal location "70". As shown in Fig. 5, first find all shortest paths from "7" (start) to "22" (location just above item), and then from "42" (location just below item) to "70" (goal).

Let $Q_{k,sp}$ denote all the states included in the calculated shortest paths. Then the total state set Q_k of robot k is

$$Q_k := Q_{k,sp} \cup \{0, 71, q_{k,item}\}. \tag{1}$$

Moreover, let $\delta_{k,sp}$ denote all the transitions included in the shortest paths; then the total transition function of robot k is given by

$$\delta_k := \delta_{k,sp} \cup \{[0, k5, q_{k,start}], [q^{\uparrow}_{k,item}, k5, q_{k,item}], [q_{k,item}, k5, q^{\downarrow}_{k,item}], [q_{k,goal}, k5, 71]\}. \tag{2}$$

In summary, the automaton model of an arbitrary robot $k(\in \{1, \dots, n\})$ is

$$\mathbf{G}_k = (Q_k, \Sigma_k, \delta_k, q_{k,0}, Q_{k,m}) \tag{3}$$

where $\Sigma_k = \{k1, k3, k5, k7\}(= \Sigma_{k,c})$, $q_{k,0} = 0$, $Q_{k,m} = \{71\}$, Q_k and δ_k are as in (1) and (2). As so defined $\mathbf{G}_k$ is nonblocking, and it is worth noting that while $\Sigma_k, q_{k,0}, Q_{k,m}$ are fixed, Q_k and δ_k depend on the given start, item, and goal locations.

3.3. Online supervisory control

We now follow the procedure introduced in Section 2 to synthesize online supervisory control for the multi-robot warehouse automation system. Since each robot's automaton $\mathbf{G}_k$ (as in (3)) has initial state $q_{k,0}$, the plant $\mathbf{G}$'s initial state is $q_0 = (q_{1,0}, \dots, q_{n,0})$. Let string s has occurred in $\mathbf{G}$ and the current state of $\mathbf{G}$ is

$$q = (q_1, \dots, q_n) := (\delta_1(q_{1,0}, s_1), \dots, \delta_n(q_{n,0}, s_n))$$

where $s_k = P_k(s)$ and $P_k : \Sigma^* \to \Sigma_k^*$ is the natural projection (Wonham and Cai, 2018). Let $N(\geq 1)$ be the size of the lookahead window.

(1) For each robot $\mathbf{G}_k$ $(k \in \{1, \dots, n\})$, generate the N-step-ahead behavior from the current state q_k by applying the truncation operation

$$\mathbf{G}_k^N(q_k) = f^N(\mathbf{G}_k, q_k).$$

Note that since $Q_{k,m} = \{71\}$ for all $k \in \{1, \dots, n\}$, $\mathbf{G}_k^N(q_k)$ does not have a marker state unless state 71 is within the N-step ahead window.

(2) Compute the plant model $\mathbf{G}^N(q)$ as the synchronous product of all truncated component models $\mathbf{G}_k^N(q_k)$:

$$\mathbf{G}^N(q) = \mathbf{G}_1^N(q_1) \| \cdots \| \mathbf{G}_n^N(q_n).$$

Note that $\mathbf{G}^N(q)$ does not have a marker state unless state 71 is within the N-step ahead window of *all* robots.

(3) Since the plant model $\mathbf{G}^N(q)$ generally does not have a marker state, we adopt for this application the optimistic attitude to make the pending states the marker states:

$$\mathbf{G}'(q) = f^N(\mathbf{G}^N(q), q) = (Q^N, \Sigma^N, \delta^N, q, Q_m'^N);$$
$$\mathbf{G}(q) = (Q^N, \Sigma^N, \delta^N, q, Q_m^N)$$

where

$$Q_m^N := Q_m'^N \cup \{q' \in Q^N \mid (\exists s \in \Sigma^*) q' = \delta^N(q, s) \,\&\, |s| = N\}.$$

(4) For control specification, we impose the safety requirement, i.e. no collisions among the robots during their navigation. Preventing collisions is equivalent to ensuring *mutual exclusion* of occupying the same cell in the grid by more than one robot. So we model this specification by an automaton $\mathbf{K}(q)$ obtained from $\mathbf{G}(q)$ by removing those states, and associated transitions, where two or more robots are in the same cell. Implicitly the optimistic attitude is taken for the specification. Moreover, we mark all states of $\mathbf{K}(q)$.

(5) Compute the supremal controllable sublanguage $\sup C(L_m(\mathbf{K}(q)))$ with respect to $\mathbf{G}(q)$, and let $\mathbf{SUP}(q) = (X^N, \Sigma^N, \xi^N, q, X_m^N)$ be the supervisor for $\mathbf{G}(q)$ that enforces $\sup C(L_m(\mathbf{K}(q)))$, i.e. $L_m(\mathbf{SUP}(q)) = \sup C(L_m(\mathbf{K}(q)))$. The control action of $\mathbf{SUP}(q)$ at state q is to enable the events in

$$\gamma(q) := \{\sigma \in \Sigma_c \mid \xi^N(q, \sigma) \in L(\mathbf{SUP}(q))\}.$$

One event $\sigma \in \gamma(q)$ will be executed. Since the robots have distinct event sets, σ belongs to one of the robot, say $\mathbf{G}_k$. So the current state $q = (q_1, \dots, q_k, \dots, q_n)$ will transition to the next state

$$q' = (q_1, \dots, \delta_k(q_k, \sigma), \dots, q_n).$$

Namely only robot $\mathbf{G}_k$ takes an action σ, and other robots stay put. The above procedure repeats with the new state q' (corresponding to the string $s\sigma$).

3.4. Case study of three robots

We demonstrate our online supervisory control scheme for warehouse automation on a concrete example of three robots with 3-step lookahead window. The start, item, and goal locations of each robot are displayed in Fig. 6. According to these locations, compute as in Section 3.2 the shortest paths; see Fig. 7. Observe that these paths are subject to multiple possibilities of collision and deadlock, which must be prevented by means of supervisory control. First, create automata $\mathbf{G}_1$, $\mathbf{G}_2$, and $\mathbf{G}_3$ for the three robots (as in (3)). These automata have state sizes 34, 25, and 25 respectively.

Consider the situation that all three robots are at their start locations. At this time, the current states of the robots are 7, 4, and 1. First, create a 3-step truncated automaton at the current state for each robot: $\mathbf{G}_1^3(7)$, $\mathbf{G}_2^3(4)$, and $\mathbf{G}_3^3(1)$, as displayed in Fig. 8. These automata have state sizes

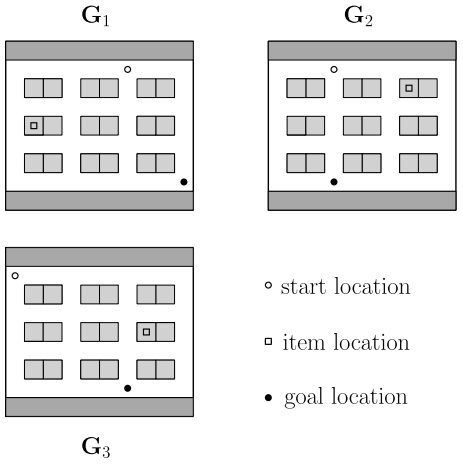

Fig. 6. Start, item, and goal locations assigned to each of the three robots.

[1] We shall consider in future work a more systematic formulation of the efficiency criterion, by first casting the problem into the timed DES framework (Brandin & Wonham, 1994) and then adapt the optimal supervisory control method (Sengupta & Lafortune, 1998) to minimize a cost function of the total time spent.

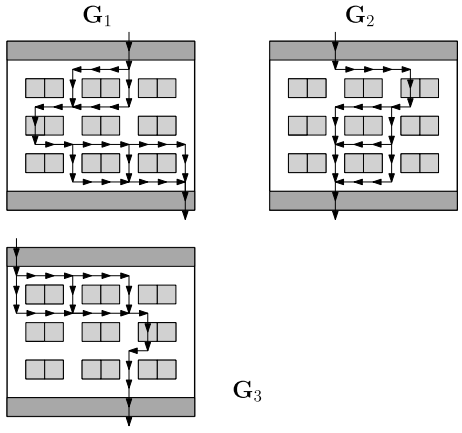

Fig. 7. Automaton models of the three robots.

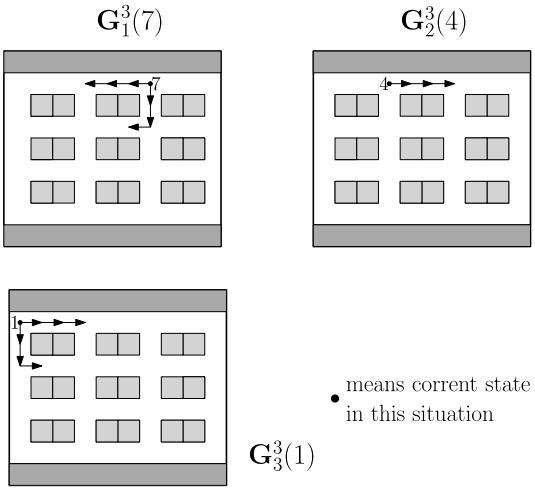

Fig. 8. The 3-step truncated automata of the robots at current states 7, 4, and 1.

7, 4, and 7; note however that none has a marker state. Then the current plant model $\mathbf{G}^3(q)$, $q = (7, 4, 1)$, is the synchronous product:

$$\mathbf{G}^3(q) = \mathbf{G}_1^3(7) \| \mathbf{G}_2^3(4) \| \mathbf{G}_3^3(1).$$

Since $\mathbf{G}_1^3(7)$, $\mathbf{G}_2^3(4)$, and $\mathbf{G}_3^3(1)$ do not have marker states, neither does $\mathbf{G}^3(q)$. Hence we adopt the optimistic attitude to make the pending states the marker states:

$$\mathbf{G}'(q) = f^3(\mathbf{G}^3(q), q) = (Q^3, \Sigma^3, \delta^3, q, Q'^3_m);$$
$$\mathbf{G}(q) = (Q^3, \Sigma^3, \delta^3, q, Q^3_m)$$

where

$$Q^3_m := Q'^3_m \cup \{q' \in Q^3 \mid (\exists s \in \Sigma^*) q' = \delta^3(q, s) \ \&\ |s| = 3\}.$$

Next, for control specification, we impose collision avoidance by mutual exclusion of occupying the same cell in the grid by more than one robot. In the current situation, at cells numbered 4, 5, 6, and 7, robot $\mathbf{G}_2$ and robot $\mathbf{G}_3$ may collide. In addition, robot $\mathbf{G}_3$ may collide with the other two robots at cell 4. So the specification automaton $\mathbf{K}(q)$ is obtained from $\mathbf{G}(q)$ by removing the states, and the associated transitions, which correspond to $(4, 4, *), (4, *, 4), (*, l, l)$ where $*$ means all states and $l \in \{4, \dots 7\}$. All states of $\mathbf{K}(q)$ are marked.

Now compute the supremal controllable sublanguage $\sup C(L_m(\mathbf{K}(q)))$ with respect to $\mathbf{G}(q)$, and let $\mathbf{SUP}(q) = (X^N, \Sigma^N, \xi^N, q, X^N_m)$ be the supervisor for $\mathbf{G}(q)$ that enforces $\sup C(L_m(\mathbf{K}(q)))$, i.e. $L_m(\mathbf{SUP}(q)) = \sup C(L_m(\mathbf{K}(q)))$. $\mathbf{SUP}(q)$ has 108 states, and its control action at state q is to enable the events in $\gamma(q) = \{15, 23, 33, 35\}$. In fact only the event 17 (robot $\mathbf{G}_1$ goes west) is disabled, because the occurrence of 17 would cause blocking between robot $\mathbf{G}_1$ and robot $\mathbf{G}_2$. One event in $\gamma(q)$ will be executed, say 33, i.e. robot $\mathbf{G}_3$ goes east. So the current state $q = (7, 4, 1)$ will transition to the next state

$$q' = (7, 4, \delta_3(1, 33)) = (7, 4, 2).$$

The online control procedure repeats with the new state q'.

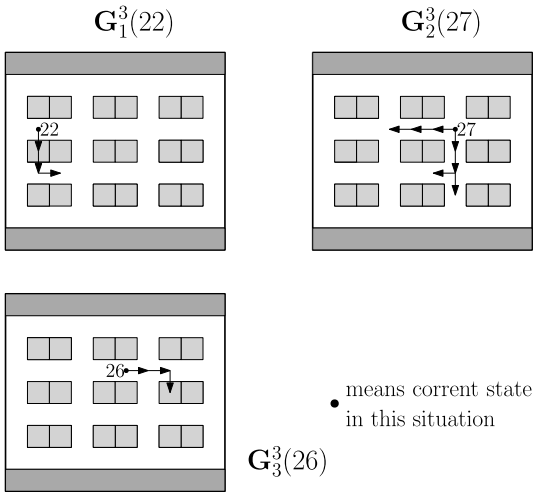

Fig. 9. The 3-step truncated automata of the robots at current states.

The above situation illustrates a supervisory control action that prevents blocking. Let us consider another situation that demonstrates a supervisory control action that avoids collision. As displayed in Fig. 9, the current states of the robots are 22, 27, and 26. First, create a 3-step truncated automaton at the current state for each robot: $\mathbf{G}_1^3(22)$, $\mathbf{G}_2^3(27)$, and $\mathbf{G}_3^3(26)$ (see Fig. 9). These automata have state sizes 4, 8, and 4; note that none has a marker state. Then the current plant model $\mathbf{G}^3(q)$, $q = (22, 27, 26)$, is the synchronous product:

$$\mathbf{G}^3(q) = \mathbf{G}_1^3(22) \| \mathbf{G}_2^3(27) \| \mathbf{G}_3^3(26).$$

Since $\mathbf{G}_1^3(22)$, $\mathbf{G}_2^3(27)$, and $\mathbf{G}_3^3(26)$ do not have marker states, neither does $\mathbf{G}^3(q)$. Hence we adopt the optimistic attitude to make the pending states the marker states:

$$\mathbf{G}'(q) = f^3(\mathbf{G}^3(q), q) = (Q^3, \Sigma^3, \delta^3, q, Q'^3_m);$$
$$\mathbf{G}(q) = (Q^3, \Sigma^3, \delta^3, q, Q^3_m)$$

where

$$Q^3_m := Q'^3_m \cup \{q' \in Q^3 \mid (\exists s \in \Sigma^*) q' = \delta^3(q, s) \ \&\ |s| = 3\}.$$

Next, for control specification, we impose collision avoidance by mutual exclusion of occupying the same cell in the grid by more than one robot. In the current situation, at cells numbered 26 and 27, robot $\mathbf{G}_2$ and robot $\mathbf{G}_3$ may collide. So the specification automaton $\mathbf{K}(q)$ is obtained from $\mathbf{G}(q)$ by removing the states, and the associated transitions, which correspond to $(*, 26, 26), (*, 27, 27)$ where $*$ means all states. All states of $\mathbf{K}(q)$ are marked.

Now compute the supremal controllable sublanguage $\sup C(L_m(\mathbf{K}(q)))$ with respect to $\mathbf{G}(q)$, and let $\mathbf{SUP}(q) = (X^N, \Sigma^N, \xi^N, q, X^N_m)$ be the supervisor for $\mathbf{G}(q)$ that enforces $\sup C(L_m(\mathbf{K}(q)))$, i.e. $L_m(\mathbf{SUP}(q)) = \sup C(L_m(\mathbf{K}(q)))$. $\mathbf{SUP}(q)$ has 68 states, and its control action at state q is to enable the events in $\gamma(q) = \{15, 25\}$. In fact both event 27 (robot $\mathbf{G}_2$ goes west) and event 33 (robot $\mathbf{G}_3$ goes east) are disabled, because the occurrence of 27 or 33 would cause collision between robot $\mathbf{G}_2$ and robot $\mathbf{G}_3$ (simultaneous occupation of cell 26 or 27). One event in $\gamma(q)$

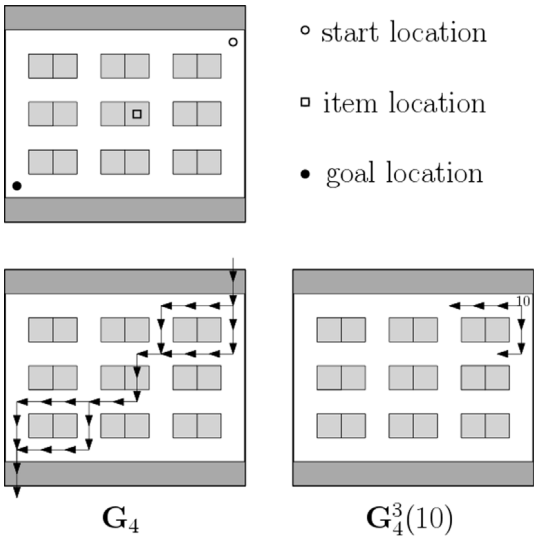

Fig. 10. Start, item, and goal locations assigned to the fourth robot; automaton model of the fourth robot; and the 3-step truncated automaton of the fourth robot at the current state 10.

will be executed, say 25, i.e. robot $\mathbf{G}_2$ goes south. So the current state $q = (22, 27, 26)$ will transition to the next state

$$q' = (22, \delta_3(27, 25), 26) = (22, 37, 26).$$

The online control procedure repeats with the new state q'.

For this case study, our online supervisory control scheme successfully guarantees that all three agents reach their goal locations without collision or blocking. Throughout the process, the largest online supervisor $\mathbf{SUP}(q)$ has state size 168. As a comparison, the offline supervisor computed by standard method in Wonham and Cai (2018) has 18,715 states; both the offline method and the online supervisory control in Chung et al. (1992, 1993, 1994) require computing the plant model $\mathbf{G}_1 \| \mathbf{G}_2 \| \mathbf{G}_3$ which has 21,250 states.

Remark 2. In addition to reducing computational effort, our supervisory control scheme is effective in addressing time-varying changes in the system. Consider again the situation in Fig. 9, and suppose that a new task has just arrived with the start, item, and goal locations displayed in Fig. 10. This new task is assigned to a new robot, the fourth robot; we create the automaton $\mathbf{G}_4$ (as in (3)) displayed in Fig. 10. The current states of the four robots are 22, 27, 26, and 10. First, we obtain a 3-step truncated automaton at the current state for each robot: $\mathbf{G}_1^3(22)$, $\mathbf{G}_2^3(27)$, $\mathbf{G}_3^3(26)$ in Fig. 9, and $\mathbf{G}_4^3(10)$ in Fig. 10 (which has 7 states). Then the current plant model $\mathbf{G}^3(q)$, $q = (22, 27, 26, 10)$, is the synchronous product:

$$\mathbf{G}^3(q) = \mathbf{G}_1^3(22) \| \mathbf{G}_2^3(27) \| \mathbf{G}_3^3(26) \| \mathbf{G}_4^3(10).$$

Since $\mathbf{G}^3(q)$ does not have marker states, we again adopt the optimistic attitude to generate $\mathbf{G}(q)$.

Next, for control specification, we again impose collision avoidance by mutual exclusion. In the current situation, the new robot $\mathbf{G}_4^3(10)$ clearly will not collide with other three robots, and hence the specification automaton $\mathbf{K}(q)$ does not change from the situation in Fig. 9. (Precisely, $\mathbf{K}(q)$ is obtained from $\mathbf{G}(q)$ by removing the states, and the associated transitions, which correspond to $(*, 26, 26, *), (*, 27, 27, *)$ where $*$ means all states.) Now compute the supremal controllable sublanguage $\sup C(L_m(\mathbf{K}(q)))$ with respect to $\mathbf{G}(q)$, and let $\mathbf{SUP}(q)$ be the supervisor for $\mathbf{G}(q)$ that enforces $\sup C(L_m(\mathbf{K}(q)))$. $\mathbf{SUP}(q)$ has 68 states, and its control action at state q is to enable the events in $\gamma(q) = \{15, 25, 45, 47\}$. One event in $\gamma(q)$ will be executed, say 45, i.e. robot $\mathbf{G}_4$ goes south. The above has shown how to deal with a newly arrived task; other time-varying changes in the system can be handled similarly, including removing a malfunctioned robot or planned paths are unexpectedly blocked (e.g. by a fallen box).

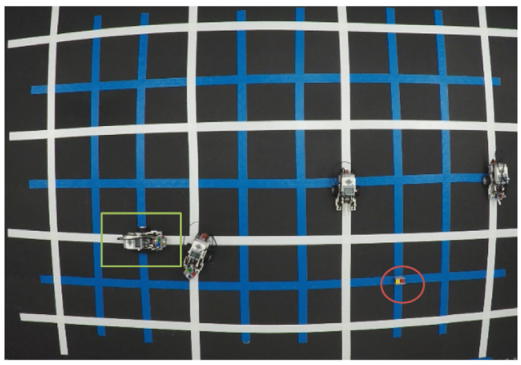

Fig. 11. Testbed layout and an experiment snapshot: four robots are moving in the area to pick up items. Only one item in the circle is left for pickup (three items have been picked up), and the leftmost robot in the box is stopped by the online supervisor to avoid collision with the nearby robot to the right. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 12. LEGO MINDSTORMS EV3 (photo by the authors).

3.5. Experiment testbed

To test the validity of our online supervisory control scheme when it is implemented on real robots, we build an experiment testbed as displayed in Fig. 11. The white lines represent passages (free space) in warehouse where robots can move, while blue lines represent item-storage space and items are placed at the intersections of blue lines.

The robot used in this testbed is the LEGO MINDSTORMS EV3 (https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313) (see Fig. 12). EV3 is a two-wheel differential-drive robot. We mount in front of the robot two color sensors, facing downward, so that the robot can trace the lines and make (left/right) turns at intersections by sensing different combinations of colors. This is the low-level continuous-time control that the robot performs autonomously. On the high-level, online supervisory control is performed. This is done by using an external laptop computer (i5 Processor and 4GB RAM): online supervisors are computed and the corresponding control actions are sent to involved robots through wireless communication (2.4 GHz).

Each intersection of two lines (either color) is treated as a state, and thus an event is a robot moving from one intersection to the next. An event is enabled or disabled by an online supervisor to ensure safety and deadlock-free; such a control action is wirelessly communicated to the corresponding robot. After each occurrence of event, a new online supervisor will be computed, and the control cycle repeats until all robots accomplish their tasks.

We have successfully tested our proposed online supervisory control scheme for four robots; a demonstration video is available in https://control.eng.osaka-cu.ac.jp/experiment.html. More extensive experiment with a larger number of robots and various different locations (start, item, goal) is being carried out.

4. Conclusions

We have presented an online supervisory control method amenable for multi-agent DES, and applied the method to controlling warehouse automation systems served by multiple mobile robots. Moreover, we have demonstrated the effectiveness of the method through a case study, as well as its validity when implemented on real robots.

In our ongoing work, we are investigating the sufficient number of lookahead steps in order to ensure nonblocking controlled behavior. We also aim to address issues of noise and bias in warehouse automation systems under a probabilistic DES framework.

Acknowledgments

This research was supported by JSPS KAKENHI Grant No. JP16K18122, and the OCU 'Think globally, act locally' Research Grant for Young Scientists 2018 through the hometown donation fund of Osaka City.

References

Boroomand, F., & Hashtrudi-Zad, S. (2013). A limited lookahead policy in robust nonblocking supervisory control of discrete event systems. In *Proc. of American control conference* (pp. 935–939).

Brandin, B., & Wonham, W. (1994). Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, *39*(2), 329–342.

Chung, S. L., Lafortune, S., & Lin, F. (1992). Limited lookahead policies in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, *37*(12), 1921–1935.

Chung, S. L., Lafortune, S., & Lin, F. (1993). Recursive computation of limited lookahead supervisory controls for discrete event systems. *Discrete Event Dynamic Systems*, *3*(1), 71–100.

Chung, S. L., Lafortune, S., & Lin, F. (1994). Supervisory control using variable lookahead policies. *Discrete Event Dynamic Systems*, *4*(3), 237–268.

Gohari, P., & Wonham, W. M. (2000). On the complexity of supervisory control design in the rw framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *30*(5), 643–652.

Goryca, J., & Hill, R. C. (2013). Formal synthesis of supervisory control software for multiple robot systems. In *Proc. of American control conference* (pp. 125–131).

Hadj-Alouane, N., Lafortune, S., & Lin, F. (1996). Centralized and distributed algorithm for on-line synthesis of maximal control policies under partial observation. *Discrete Event Dynamic Systems*, *6*(4), 379–427.

Hill, R. C., & Lafortune, S. (2016). Planning under abstraction within a supervisory control context. In *Proc. of the 55th IEEE conference on decision and control* (pp. 4770–4777).

Hill, R. C., & Lafortune, S. (2017). Scaling the formal synthesis of supervisory control software for multiple robot systems. In *Proc. of American control conference* (pp. 3840–3847).

Kumar, R., Cheung, H. M., & Marcus, S. I. (1998). Extension based limited lookahead supervision of discrete event systems. *Automatica*, *34*(11), 1327–1344.

Lopes, Y. K., Trenkwalder, S. M., Leal, A. B., Dodd, T. J., & Groß, R. (2016). Supervisory control theory applied to swarm robotics. *Swarm Intelligence*, *10*(1), 65–97.

Ramadge, P. J., & Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, *25*(1), 206–230.

Sengupta, R., & Lafortune, S. (1998). An optimal control theory for discrete event systems. *SIAM Journal on Control and Optimization*, *36*(2), 488–541.

Su, R. (2013). Discrete-event modeling of multi-agent systems with broadcasting-based parallel composition. *Automatica*, *49*(11), 3502–3506.

Su, R., & Lennartson, B. (2017). Control protocol synthesis for multi-agent systems with similar actions instantiated from agent and requirement templates. *Automatica*, *79*(5), 244–255.

Su, R., & Lin, L. (2013). Synthesis of control protocols for multi-agent systems with similar actions. In *Proc. of the 52nd IEEE conference decision control* (pp. 147–152).

Ware, S., & Su, R. (2016). Incremental scheduling of discrete event systems. In *Proc. of the 13th international workshop on discrete event systems* (pp. 147–152).

Winacott, C., & Rudie, K. (2009). Limited lookahead supervisory control of probabilistic discrete-event systems. In *Proc. of the 47th annual allerton conf.* (pp. 660–667).

Wonham, W. M., & Cai, K. (2018). *Supervisory control of discrete-event systems. Communications and control engineering*. Springer.

Wonham, W. M., Cai, K., & Rudie, K. (2018). Supervisory control of discrete-event systems: a brief history. *Annual Reviews in Control*, *45*, 250–256.

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, *29*(1), 9–19.