「システム設計への事象駆動アプローチ—応用と展開」特集号 解　説

Application of Supervisory Control Theory with Warehouse Automation Case Study*

Yuta TATSUMOTO†, Masahiro SHIRAISHI† and Kai CAI†

1. Introduction

Supervisory control theory of discrete-event systems was first proposed by Ramadge and Wonham in the 1980s [3], with the aim to formalizing general (high-level) control principles for a wide range of application domains. In this theory, discrete-event systems 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. This feature leads to the fundamental concept of *language controllability*, which determines the existence of a supervisor that dynamically disables suitably selected events in order to satisfy an imposed control specification. For a comprehensive account of supervisory control theory, the reader is referred to [7]; also see [8] for a recent historical overview of the theory.

Over the past three decades, many applications of supervisory control have been proposed in areas such as flexible manufacturing, communication protocols, transportation systems, database management, and logistic services. Several remarkable applications are: (1) the testbed assembly process of the Atelier Interétablissement Productique (AIP) in Grenoble, France [1], (2) the design of a telephone directory assistance call center[4], and (3) the patient support system for a magnetic resonance image (MRI) scanner [5].

In this paper we provide a tutorial on supervisory control theory, and introduce a new application on warehouse automation using mobile robots. In presenting supervisory control theory, rather than focusing on mathematical details, we demonstrate the procedure of supervisory control design by using an easy-to-use software package *TCT* [6]. Our purpose is for control scientists and engineers who may not be familiar with the theory to still carry out the design of supervisory controllers step-by-step in *TCT*.

The new application to be introduced on warehouse automation has received significant interest from both academia and industries, due to its central role in the rapidly growing e-commerce, supply chain, and material handling enterprise. A landmark of such examples is the Kiva System [9], which dispatches hundreds of robots to serve the logistics in Amazon's distribution centers. Not only being a critical technology to achieving unprecedented logistic efficiency, warehouse automation is also expected to provide a solution to the labor shortage problem owing to population decline in many developed countries. For this important application, we show how to model and control warehouses served by multiple robots using supervisory control theory and design software *TCT*.

2. Supervisor Control Theory and Design Package TCT

In supervisory control [3,7], the plant to be controlled is modeled by a finite-state *automaton* $\mathbf{G}$

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

where Q is the finite state set, $q_0 \in Q$ the *initial state*, $Q_m \subseteq Q$ the set of *marker states* (or target states), Σ the finite *event set*, and $\delta : Q \times \Sigma \to Q$ the (partial) *state transition function*. Let Σ^* be the set of all finite-length sequences of events in Σ, and extend δ such that $\delta : Q \times \Sigma^* \to Q$. We write $\delta(q,s)!$ to mean that $\delta(q,s)$ is defined. The event set Σ is partitioned into a subset Σ_c of *controllable* events and a subset Σ_u of *uncontrollable* events; only controllable events can be enabled or disabled by an external entity, called *supervisor*, introduced below.

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.

In the design package *TCT* [6], an automaton $\mathbf{G}$ is created by the procedure **create**. In response to the prompt, the user enters the automaton name, number of states n, the list of marker states and list of transitions (of the form [exit state, event, entering state]). The *TCT* standard state set is the integer set $\{0,1,...,n-1\}$, with 0 the initial state. Event labels

* This work was supported in part by JSPS KAKENHI Grant no. JP16K18122 and OCU 'Think Globally, Act Locally' Research Grant for Young Scientists 2018 through the hometown donation fund of Osaka City.

† Department of Electrical and Information Engineering, Osaka City University

Key Words: supervisory control, discrete-event systems, automata, multi-robot warehouse automation.

are integers between 0 and 999, where odd numbers represent controllable events and even numbers represent uncontrollable events. We write this procedure as follows:

$$\mathbf{G} = \mathbf{create}(\text{G, [marker states],[transitions]}) \ (n,m).$$

where n is the number of states and m the number of transitions.

To display the content of the created automaton **G**, use procedure **SE**. To modify **G** use procedure **edit**. Moreover, procedure **CE** converts **G** to a GIF file, which may be displayed graphically by procedure **DE** as a labeled transition graph; marker states are represented by a double circle. An additional procedure **allevents**, which will be useful below, extracts all the events of an automaton; specifically,

$$\mathbf{ALLG} = \mathbf{allevents}(\mathbf{G})$$

generates a marked one-state automaton self-looped with all the events of **G**.

When the plant consists of multiple components (as in the case study to be introduced in Section **3**), the plant model **G** is formed by taking the *synchronous product* of the components $\mathbf{G}_1,\ldots,\mathbf{G}_N$. In *TCT* this is done by procedure **sync**:

$$\mathbf{G} = \mathbf{sync}(\mathbf{G_1},\ldots,\mathbf{G_N}).$$

In case $\mathbf{G_1},\ldots,\mathbf{G_N}$ have the same event set, then **sync** may be replaced by procedure **meet**.

In supervisory control a control specification is also modeled by an automaton **E** (say). When there are multiple specifications $\mathbf{E}_1,\ldots,\mathbf{E}_M$, the overall **E** is (again) the synchronous product:

$$\mathbf{E} = \mathbf{sync}(\mathbf{E_1},\ldots,\mathbf{E_M}).$$

To ensure that **E** is defined on the same event set as the plant **G** (which is important for supervisor synthesis below), compute the synchronous product of **E** and **ALLG** that has all events of **G**:

$$\mathbf{SPEC} = \mathbf{sync}(\mathbf{E},\mathbf{ALLG}).$$

Given a plant **G** and a specification **SPEC**, one may proceed to synthesize a supervisor for **G** to enforce **SPEC**. For this the key concept is *language controllability*. Informally, controllability of $L_m(\mathbf{SPEC})$ (with respect to **G**) means that no strings in $L(\mathbf{SPEC})$ can exit $L(\mathbf{SPEC})$ when followed by an uncontrollable event in **G**. Precisely, $L_m(\mathbf{SPEC})$ is controllable if for every $s \in \Sigma^*$ and every $\sigma \in \Sigma$,

$$s \in L(\mathbf{SPEC}), \sigma \in \Sigma_u, s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in L(\mathbf{SPEC}).$$

Controllability is proved to be closed under set unions, and therefore a unique supremal controllable sublanguage of a given language exists.

In *TCT*, procedure **supcon** computes an *optimal and nonblocking* supervisor for plant **G** to enforce specification **SPEC** as follows:

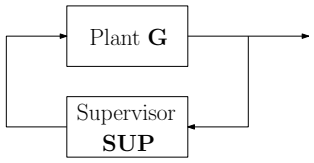


Fig. 1 Supervisory control implementation

$$\mathbf{SUP} = \mathbf{supcon}(\mathbf{G},\mathbf{SPEC}).$$

SUP is a nonblocking automaton, i.e. $L(\mathbf{SUP}) = \overline{L_m(\mathbf{SUP})}$. Moreover, $L_m(\mathbf{SUP})$ is the supremal controllable sublanguage of $L_m(\mathbf{G}) \cap L_m(\mathbf{SPEC})$; in this sense **SUP** is optimal. This optimality, interpreted in terms of controlled behavior, means maximally permissive (or minimally restrictive) behavior. Procedure **condat** generates a table that lists the (controllable) events to be disabled at each state of **SUP**:

$$\mathbf{SUP}.\text{DAT} = \mathbf{condat}(\mathbf{G},\mathbf{SUP}).$$

To display **SUP**.DAT, use procedure **SA**. The implementation of supervisor **SUP** is the familiar state-feedback loop, as displayed in Fig. 1. **SUP** monitors the current state of plant **G** (via 'observing' a corresponding string), and decides the (controllable) events to be disabled according to the **condat** table.

It is often convenient to simplify a computed supervisor **SUP**; for this apply procedure **supreduce** as follows:

$$\mathbf{SIMSUP} = \mathbf{supreduce}(\mathbf{G},\mathbf{SUP},\mathbf{SUP}.\text{DAT}).$$

The simplified supervisor satisfies

$$\mathbf{sync}(\mathbf{G},\mathbf{SIMSUP}) = \mathbf{SUP}.$$

Finally to create a purely distributed control architecture, and this is of particular interest for the case study introduced below, *TCT* procedure **localize** effectively decomposes supervisor **SUP** into a set of *local controllers*, one for each plant component:

$$\{\mathbf{LOC_1},\ldots,\mathbf{LOC_N}\} = \mathbf{localize}(\mathbf{G},\mathbf{SUP},\mathbf{G_1},\ldots,\mathbf{G_N})$$

Moreover, it is guaranteed [2] that the collective local controlled behavior is identical to the global behavior, i.e.

$$\mathbf{sync}(\mathbf{G_1},\mathbf{LOC_1},\ldots,\mathbf{G_N},\mathbf{LOC_N}) = \mathbf{SUP}.$$

3. Case Study: Warehouse Automation using Multiple Robots

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 system [9]. In this section we show how to apply supervisory control theory and software *TCT* 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. 2. Mobile robots are

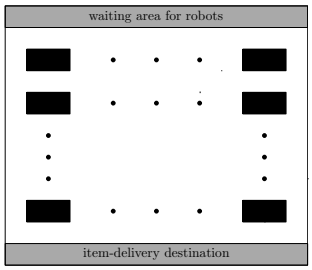


Fig. 2 Warehouse configuration: items to be picked up are stored in black-rectangle areas.

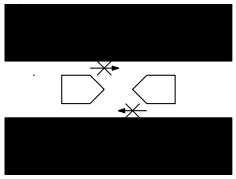


Fig. 3 Deadlock (or blocking): neither robots can move forward and consequently they cannot finish their delivery tasks.

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.

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. 3) such that no robot 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 supervisory control

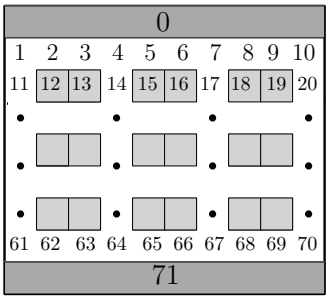


Fig. 4 Warehouse grid assigned with numbers

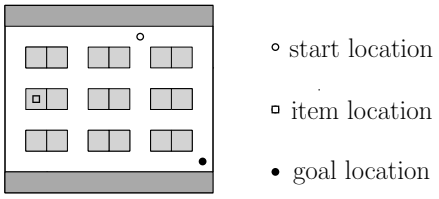


Fig. 5 Example: start, item, and goal locations assigned to a robot

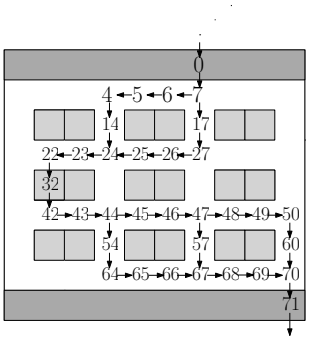


Fig. 6 Automaton model of the robot in Fig. 5

theory 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 individual robots.

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. 4. Assign natural numbers to each cell (i.e. location) 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 (odd numbers in TCT).

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

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

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. As an example, in Fig. 5 a robot is assigned a start location "7", item location "32", and goal location "70". As shown in Fig. 6, 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 the set of all 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 the set of all 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 $k(\in \{1,...,N\})$ 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 Supervisor Synthesis by TCT

We now follow the recipe introduced in Section **2** to design an optimal and nonblocking supervisor by *TCT*. First, use procedure **create** to create an automaton $\mathbf{G_k}$ (as in (3)) for each $k \in \{1,...,N\}$. These are treated as plant components. Then compute the total plant $\mathbf{G}$ to be controlled by

$$\mathbf{G} = \mathbf{sync}(\mathbf{G_1},...,\mathbf{G_N}).$$

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. For this we use *TCT* procedure **mutex**. Specifically

$$\mathbf{E_{ij}} = \mathbf{mutex}(\mathbf{G_i},\mathbf{G_j},[[1,1],...,[70,70]])$$

is formed from the synchronous product of $\mathbf{G_i}$ and $\mathbf{G_j}$, by excluding state pairs [1,1],...,[70,70]. Now form $\mathbf{E_{ij}}$ for all pairs of robots $i,j \in \{1,...,N\}$ with $i<j$; the total specification $\mathbf{E}$ is

$$\mathbf{E} = \mathbf{sync}(\mathbf{E_{12}},...,\mathbf{E_{(N-1)N}}).$$

To ensure that $\mathbf{E}$ is defined on the same event set as plant $\mathbf{G}$, form

$$\mathbf{SPEC} = \mathbf{sync}(\mathbf{E},\mathbf{ALLG})$$

where $\mathbf{ALLG} = \mathbf{allevents}(\mathbf{G})$.

Having obtained the plant $\mathbf{G}$ and specification $\mathbf{SPEC}$, we apply procedure **supcon** to compute the optimal and nonblocking supervisor

$$\mathbf{SUP} = \mathbf{supcon}(\mathbf{G},\mathbf{SPEC}).$$

First, $\mathbf{SUP}$ enforces specification $\mathbf{E}$, namely guarantees the safety requirement. Second, $\mathbf{SUP}$ is nonblocking, i.e. deadlocks as described in Fig. 2 are prevented. Except enforcing safe and deadlock-free controlled behavior, $\mathbf{SUP}$ places no further constraint on the robots' behavior; in this sense $\mathbf{SUP}$ is maximally permissive.

To check the control actions of $\mathbf{SUP}$, namely which events are disabled at which state of $\mathbf{SUP}$, compute

$$\mathbf{SUP}.\mathrm{DAT} = \mathbf{condat}(\mathbf{G},\mathbf{SUP}).$$

$\mathbf{SUP}$.DAT facilitates the standard state-feedback implementation of supervisor $\mathbf{SUP}$ for plant $\mathbf{G}$ (as in Fig. 1). To obtain a more compact supervisor with fewer number of states, apply

$$\mathbf{SIMSUP} = \mathbf{supreduce}(\mathbf{G},\mathbf{SUP},\mathbf{SUP}.\mathrm{DAT}).$$

Finally, for the multi-robot plant it may be of particular interest to obtain local controllers for the individual agents, thereby making the robots autonomous. This is achieved by procedure **localize**, as follows

$$\{\mathbf{LOC_1},...,\mathbf{LOC_N}\} = \mathbf{localize}(\mathbf{G},\mathbf{SUP},\mathbf{G_1},...,\mathbf{G_N})$$

which systematically decomposes the *global* supervisor $\mathbf{SUP}$, *external* to the robots, into *local* control strategies *internal* to the relevant robots. With each robot $\mathbf{G_k}$ $(k \in \{1,...,N\})$ equipped with its own local controller $\mathbf{LOC_k}$, the original *centralized* control architecture is transformed into a purely *distributed* one.

3.4 Example of Three Robots

We demonstrate our approach to warehouse automation on a concrete example of three robots. The start, item, and goal locations of each robot are displayed in Fig. 7. According to these locations, compute as in Section **3.2** the shortest paths; see Fig. 8. Observe that these paths present multiple possibilities of collision and deadlock, which must be prevented by means of supervisory control.

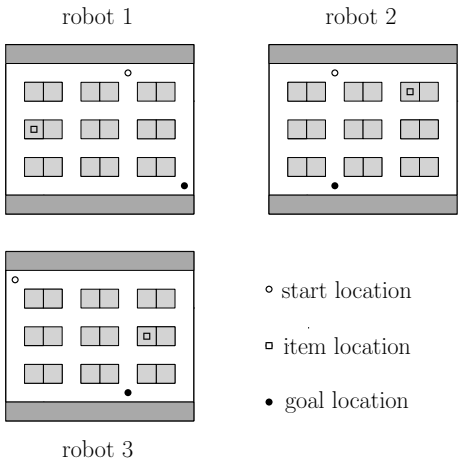


Fig. 7 Start, item, and goal locations assigned to each of the three robots

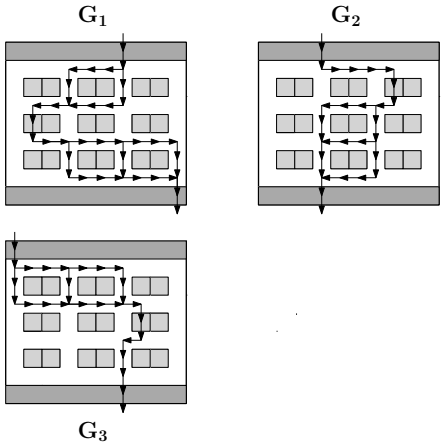


Fig. 8 Automata models of the three robots

First, use procedure **create** to create (as in (3)) automata $\mathbf{G_1}$, $\mathbf{G_2}$, and $\mathbf{G_3}$ for the three robots. These automata have state sizes 34, 25, and 25 respectively. Thus the total plant $\mathbf{G}$ to be controlled is

$$\mathbf{G} = \mathbf{sync}(\mathbf{G_1},\mathbf{G_2},\mathbf{G_3}) \text{ (21250,66700)}$$

(21250,66700) means that $\mathbf{G}$ has 21250 states and 66700 transitions.

Next for control specification, we impose collision-avoidance by procedure **mutex** as follows:

$$\mathbf{E_{12}} = \mathbf{mutex}(\mathbf{G_1},\mathbf{G_2},[[1,1],\ldots,[70,70]]) \text{ (812,1661)}$$
$$\mathbf{E_{13}} = \mathbf{mutex}(\mathbf{G_1},\mathbf{G_3},[[1,1],\ldots,[70,70]]) \text{ (809,1658)}$$
$$\mathbf{E_{23}} = \mathbf{mutex}(\mathbf{G_2},\mathbf{G_3},[[1,1],\ldots,[70,70]]) \text{ (605,1227)}$$

Hence the total specification $\mathbf{E}$ is

$$\mathbf{E} = \mathbf{sync}(\mathbf{E_{12}},\mathbf{E_{13}},\mathbf{E_{23}}) \text{ (18715,55870)}$$

Moreover compute

$$\mathbf{ALLG} = \mathbf{allevents}(\mathbf{G}) \text{ (1,9)}$$
$$\mathbf{SPEC} = \mathbf{sync}(\mathbf{E},\mathbf{ALLG}) \text{ (18715,55870)}$$

such that **SPEC** is defined on the same event set as plant $\mathbf{G}$.

Now synthesize the optimal and nonblocking supervisor by

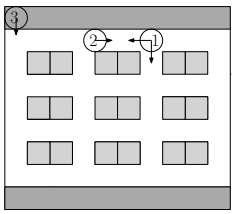


Fig. 9 Supervisory control logic: prevent deadlock between robots 1 and 2

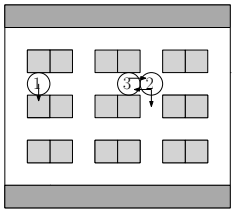


Fig. 10 Supervisory control logic: prevent collision between robots 2 and 3

$$\mathbf{SUP} = \mathbf{supcon}(\mathbf{G},\mathbf{SPEC}) \text{ (18715,55870)}$$

SUP achieves safe, deadlock-free, and maximally permissive controlled behavior. The control actions of **SUP** may be inspected by looking up in

$$\mathbf{SUP}.\mathrm{DAT} = \mathbf{condat}(\mathbf{G},\mathbf{SUP}).$$

Here we explain **SUP**'s control logic in two scenarios, one to prevent deadlock and the other to prevent collision.

(1) As displayed in Fig. 9, while robot $\mathbf{G_2}$ can only go east, robot $\mathbf{G_1}$ has two choices of either go west or go south. In this scenario, supervisor **SUP** disables the event that robot $\mathbf{G_1}$ goes west, for otherwise robots $\mathbf{G_1}$ and $\mathbf{G_2}$ would block each other in the aisle and a deadlock would consequently occur.

(2) As displayed in Fig. 10, while robot $\mathbf{G_3}$ can only go east, robot $\mathbf{G_2}$ has two choices of either go west or go south. In this scenario, supervisor **SUP** disables two events – robot $\mathbf{G_3}$ goes east and robot $\mathbf{G_2}$ goes west – to prevent collision. As a result, the only possible event for these two robots at this scene is that $\mathbf{G_2}$ goes south.

Since the supervisor **SUP** has a large number of states, it is reasonable to try reducing its state size by

$$\mathbf{SIMSUP} = \mathbf{supreduce}(\mathbf{G},\mathbf{SUP},\mathbf{SUP}.DAT)(6421,35585)$$

Observe that the state size of **SIMSUP** is roughly one third of that of **SUP**.

Finally, to create a purely distributed control architecture for the three robots where each robot is equipped with its own local controller, we apply the procedure **localize** as follows

$$\{\mathbf{LOC_1},\mathbf{LOC_2},\mathbf{LOC_3}\} = \mathbf{localize}(\mathbf{G},\mathbf{SUP},\mathbf{G_1},\mathbf{G_2},\mathbf{G_3})$$

These local controllers have state sizes 6639, 4739,

and 5929 respectively. It is moreover verified that the collective local controlled behavior is identical to that achieved by the global supervisor **SUP**:

$$\mathbf{sync}(\mathbf{G_1},\mathbf{LOC_1},\mathbf{G_2},\mathbf{LOC_2},\mathbf{G_3},\mathbf{LOC_3}) = \mathbf{SUP}.$$

Equipped with individual local controllers each robot becomes capable of making its own control decisions based on observation and/or communication with peers.

4. Conclusions

We have provided a tutorial on supervisory control theory, in particular the control design procedure by using software package *TCT*. Moreover, we have demonstrated the steps of applying the theory and software to model and control warehouse automation using mobile robots.

While supervisory control can provide an effective solution to warehouse automation in principle, the curse is dimensionality (as is the case for many theoretically-sound methods). With the increase in the size of warehouses and the number of robots, the computation of the optimal supervisor may become practically infeasible. To tackle this issue, more advanced tools and algorithms await to be developed; several promising directions of research include decentralized/hierarchical architectures, online look-ahead strategies, efficient symbolic computation (e.g. using binary decision diagrams), and synergy with AlphaGo-type AI learning.

(2017 年 11 月 27 日受付)

References

[1] B. Brandin and F. Charbonnier: The supervisory control of the automated manufacturing system of the AIP; *Proc. Fourth International Conf. on Computer Integrated Manufacturing and Automation Technology*, pp. 319–324 (1994)

[2] K. Cai and W. M. Wonham: Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems; Lecture Notes in Control and Information Sciences, Vol. 459, Springer (2016)

[3] P. J. Ramadge and W. M. Wonham: Supervisory control of a class of discrete event processes; *SIAM J. Control and Optimization*, Vol. 25, No. 1, pp. 206–230 (1987)

[4] M. Seidl: Systematic controller design to drive high-load call centers; *IEEE Trans. Control Systems Technology*, Vol.14, No. 2, pp. 216–223 (2006)

[5] R. Theunissen, M. Petreczky, R. Schiffelers, D. van Beek and J. Rooda: Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner; *IEEE Trans. Automation Science and Engineering*, Vol. 11, No. 1, pp. 20–32 (2013)

[6] W. M. Wonham: Design software: XPTCT. Systems Control Group, Dept. of Electrical and Computer Engineering, University of Toronto, updated annually 1998–2017. Available online at http://www.control.toronto.edu/DES (2017)

[7] W. M. Wonham and K. Cai: Supervisory control of discrete-event systems; *Communications and Control Engineering*, Springer (2018)

[8] W. M. Wonham, K. Cai and K. Rudie: Supervisory control of discrete-event systems: a brief history – 1980-2015; *Proc. of the 20th IFAC World Congress*, pp. 1827–1833 (2017)

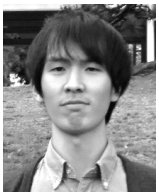
[9] P. R. Wurman, R. D'Andrea and M. Mountz: Coordinating hundreds of cooperative, autonomous vehicles in warehouses; *AI Magazine*, Vol. 29, No. 1, pp. 9–19 (2008)

Authors

辰本 佑太（たつもと ゆうた）

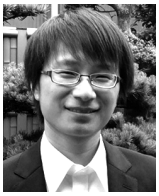
2017 年大阪市立大学工学部電気情報工学科卒業. 現在同大学大学院工学研究科電子情報系専攻前期博士課程に在学中. 離散事象マルチエージェントシステムについて研究を行っている.

白石 昌大（しらいし まさひろ）



2017 年大阪市立大学工学部電気情報工学科卒業. 現在同大学大学院工学研究科電子情報系専攻前期博士課程に在学中. 離散事象マルチエージェントシステムについて研究を行っている.

Kai Cai



Kai Cai received the B. Eng. degree in Electrical Engineering from Zhejiang University (China) in 2006, the M.A.Sc. degree in Electrical and Computer Engineering from the University of Toronto (Canada) in 2008, and the Ph.D. degree in Systems Science from Tokyo Institute of Technology (Japan) in 2011. He is currently an Associate Professor in Osaka City University; preceding this position he was an Assistant Professor in the University of Tokyo (2013-2014), and a Postdoctoral Fellow in the University of Toronto (2011-2013). Dr. Cai's research interests include distributed control of discrete-event systems and cooperative control of networked multiagent systems. He is the co-author (with W.M. Wonham) of Supervisor Localization (Springer 2016). He received the Best Paper Award of SCIE in 2013, the Best Student Paper Award of the IEEE Multi-Conference on Systems and Control and the Young Author's Award of SCIE in 2010. He is serving as an Associate Editor for the IEEE Transactions on Automatic Control.