2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Supervisory Control of Robot Swarms Using Public Events

Yuri Kaszubowski Lopes[1], Stefan M. Trenkwalder[2], André B. Leal[3],
Tony J. Dodd[4] and Roderich Groß[2]

Abstract— **Supervisory Control Theory (SCT) provides a formal framework for controlling discrete event systems. It has recently been used to generate correct-by-construction controllers for swarm robotics systems. Current SCT frameworks are limited, as they support only (private) events that are observable within the same robot. In this paper, we propose an extended SCT framework that incorporates (public) events that are shared among robots. The extended framework allows to model formally the interactions among the robots. It is evaluated using a case study, where a group of mobile robots need to synchronise their movements in space and time—a requirement that is specified at the formal level. We validate our approach through experiments with groups of e-puck robots.**

I. INTRODUCTION

Swarm robotics comprise numerous, relatively homogeneous robots that cooperate to accomplish goals. They are characterised by a scalable design, limited communication, and robust, emergent behaviour. Such properties would match the requirements of many real-world applications [1]. However, most swarm robotics systems use controllers that have been developed in an ad-hoc manner. The resulting control software is typically difficult to comprehend, making it non-trivial to perform any validation or maintenance.

Supervisory Control Theory (SCT) is a formal approach to designing and synthesising controllers [2], [3]. It assumes a discrete event system (DES) [4]. SCT uses formal languages to model how the state of the system can possibly evolve (i.e. a capability), and how it should evolve (i.e. a specification). From these languages, a controller called supervisor is synthesised, which is guaranteed to be minimally restrictive and non-blocking [2], [3]. This means that the state evolution of the system is only restricted to prevent present or future violations of the specification; moreover, it must be possible for the system to settle in a desired subset of states in the future.

[1]Y. K. Lopes is with Department of Software Engineering, Federal University of Technology, Paraná, Brazil yurilopes@utfpr.edu.br

[2]S. M. Trenkwalder and R. Groß are with Sheffield Robotics, The University of Sheffield, Sheffield, United Kingdom {s.trenkwalder, r.gross}@sheffield.ac.uk

[3]A. B. Leal is with the Department of Electrical Engineering, Santa Catarina State University, Campus Universitário Prof. Avelino Marcante, Joinville, Brazil andre.leal@udesc.br

[4]T. J. Dodd is with the School of Creative Arts and Engineering, Staffordshire University, Stoke-on-Trent, UK tony.dodd@staffs.ac.uk

Initial studies were primarily concerned with the theoretical aspects of supervisory control [5]. In later studies, the implementation aspects have gained increasing attention. A large body of work has been concerned with SCT implementations for manufacturing systems [6], [7], which are still commonly equipped with programmable logic controllers (PLC). The implementations on PLCs differ from those on microcontrollers, the latter being the common architecture in swarm robotics. In swarms of robots, additional challenges arise, such as the system being distributed, mobile and subject to significant constraints and uncertainties regarding sensing, actuation and communication.

Previous studies explored how SCT can be extended to incorporate the requirements of swarm robotics. In [8], it was shown how SCT can be used to automatically generate the control software for a swarm of 600 robots that had to organise into functional groups. In [9], a scenario was investigated where a group of agents needed to accomplish a set of spatial tasks. A global planner was used to propose optimal task allocations. In [10], probabilistic cooperative behaviours were investigated, and deployed on physical robots. In [11], SCT was used to model a set of behaviours for a group of simulated robots performing navigation tasks in an environment with obstacles. A centralized scheduling agent was used.

Communication *among robots*—an important property of swarm robotics (and multi-robot systems)—has not yet been formally defined with SCT frameworks. In this paper, we introduce the concept of *public* and *private* events to address this issue. A private event is locally dealt with by the robot in which it occurs, as it is common in SCT frameworks. In contrast, public events are communicated to other robots of the group, and hence observable beyond the boundaries of individual robots. They allow a formal modelling of communication. The resulting models[1] are then used to synthesise automatically a supervisor, and the associated control software to be deployed on the physical robots.

II. SYNCHRONOUS MOVEMENT CASE STUDY

To illustrate the use of public events, consider a group of robots (swarm) that operate in a bounded 2-D

[1]Although the models we use to describe the system and its specification are designed manually by an expert, they could in principle be obtained through automatic design approaches [12].

environment (arena). The robots are autonomous, move using differential wheels, and have obstacle sensors. All robots have to perform concurrently the same sequence of basic movements, comprising of (i) moving forward, (ii) turning left, and (iii) turning right. The order of movements is agreed at run time. Robots that detect obstacles do not perform the agreed movement, but instead perform one of two collision avoidance movements: (i) rotating left or (ii) rotating right.

This paper focuses on the integration of public events into the SCT framework. We assume that each robot can broadcast simple signals to the rest of the group. Where this is not possible, routing algorithms such as flooding could be used.

III. SUPERVISORY CONTROL THEORY

In SCT, a set of events, Σ, governs the evolution of the system. It consists of two disjoint subsets: uncontrollable events, Σ_u, and controllable events, Σ_c. Uncontrollable events represent the control input, for example, feedback signals from sensors or a timer's set off. Controllable events represent the control output, for example, a control signal to move a robot or activate a timer.

In SCT, *free behaviour models* define what the system can do, and *control specifications* define what it should do. Free behaviour models describe the capabilities of the robot, each component of the robot (e.g. a wheel, a sensor) can be modelled separately. The control specifications describe the desired behaviour of the system. For example, a robot is allowed to move forward if no obstacle has been detected ahead. Free behaviour models and control specifications are then used to calculate a control agent called *supervisor* by a process called synthesis [4].

A. Generators

There are multiple formal representations of DES (e.g. Petri-nets [13], Markov chains [14]). We use generators [8] as they resemble the deterministic finite automaton (DFA), which is commonly used to represent controllers of swarm robotics systems. While a DFA recognises whether a given word belongs to a regular language, a generator produces words of that language. Formally, a generator is a 5-tuple,

$$G = (Q, \Sigma, \delta, q_0, Q_m), \tag{1}$$

where Q is a finite set of states; Σ is a finite set of events; $\delta : Q \times \Sigma \to Q$ is a partial transition function; $q_0 \in Q$ is the initial state; and $Q_m \subseteq Q$ is a set of marked states[2]. We employ several generators—each one with their own states—splitting the design in concise models. The generators can represent both free behaviour models (describing all of the robot's capabilities) and control specifications (restricting these capabilities to a desired behaviour).

[2]Marked states are states that are considered safe for the system (e.g. they could correspond to the end of a task).

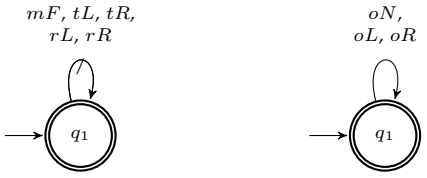


Fig. 1. Free behaviour models for random movement with collision avoidance. G_1 (left) represents the movement, G_2 (right) represents the sensing.

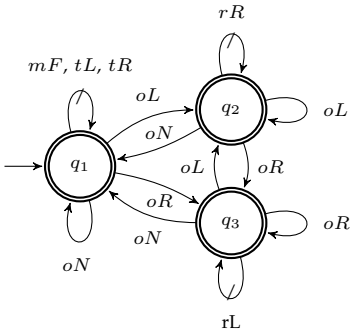


Fig. 2. Control specification, E_1, for random movement with collision avoidance.

B. Free Behaviour Models

In this section, we focus on the free behavior models that a robot needs when acting in isolation (i.e. group size 1). They are represented using generators, as shown in Figure 1. Free behaviour model G_1 defines the movements that the robot can choose from. The controllable events mF, tL, and tR choose a basic movement: move forward, turn left, or turn right, respectively. The controllable events rL and rR choose a collision avoidance movement: rotate (in place) counter-clockwise or clockwise, respectively. The chosen movement is executed until a new event occurs. Free behaviour model G_2 defines the robot's ability to detect nearby obstacles (i.e. other robots or walls) to the left, in front of, or to the right of the robot. If no obstacle is detected, uncontrollable event oN occurs. Otherwise, uncontrollable event oL or oR occur, indicating whether the obstacle(s) is/are perceived more to the left or to the right side of the robot. If the situation is symmetric (e.g. a robot in the front), the obstacles are assumed to be on the right side. Every $0.1\,\mathrm{s}$, one of oN, oL, oR occurs.

C. Control Specifications

This section presents the control specification E_1 that a robot needs when acting in isolation. It is represented using a generator in Figure 2. In state q_1, no obstacle is detected (i.e. oN). The robot can choose any basic movement (i.e. mF, tL, tR). Note that when multiple controllable events are permissible, it is common for the implementation layer to choose at random and uniformly, which in this case results in a random walk. If an object is detected on the left or right side the active

state is q_2 or q_3, respectively. In state q_2 the robot can only rotate to the right (i.e. rR) to avoid a collision with the object to the left. Similarly, in state q_3 the robot can only rotate to the left (i.e. rL).

IV. PUBLIC EVENTS

Private events are sufficient if we have only a single robot performing the task, as illustrated in Section III. In the presence of multiple robots, however, the robots need to share information to reach agreement on what movements to perform. For this purpose, we extend the SCT framework by the concept of public events.

A generator with public events is a 7-tuple,

$$G = (Q, \Sigma, \delta, q_0, Q_m, \Sigma^{ext}_{u,pub}, M), \tag{2}$$

where

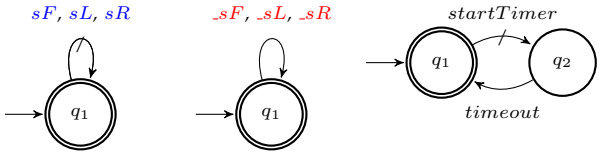
- Σ is a finite set comprising *private uncontrollable* events, $\Sigma_{u,priv}$; *private controllable* events, $\Sigma_{c,priv}$; *public uncontrollable* events, $\Sigma_{u,pub}$; and *public controllable* events, $\Sigma_{c,pub}$; where $\Sigma_{u,priv}$, $\Sigma_{c,priv}$, $\Sigma_{u,pub}$, and $\Sigma_{c,pub}$ are pairwise disjoint sets;
- $\Sigma^{ext}_{u,pub}$ is the set that comprises the public uncontrollable events, $\Sigma_{u,pub}$, of G and those of all other generators in use (if any);
- $M : \Sigma_{c,pub} \to \Sigma^{ext}_{u,pub}$ is a function that maps a public controllable event onto a public uncontrollable event;
- Q, δ, q_0, and Q_m are defined as in (1).

Private uncontrollable events are confined to the robot within which they are triggered. They are commonly used in SCT. When a public uncontrollable event, $e_{u,pub} \in \Sigma_{u,pub}$, occurs within a robot, any other robot is notified, and then evolves its current supervisor state as $q_{k+1} = \delta(q_k, e_{u,pub})$.

Note that controllable events represent potential choices for the internal controller of a specific robot. They are not supposed to control directly the operation of other robots. Hence, prior to being transmitted via the network, public controllable events are converted into public uncontrollable events by $M : \Sigma_{c,pub} \to \Sigma^{ext}_{u,pub}$. In other words, a public controllable event, $e_{c,pub}$, evolves another robot's state as $q_{k+1} = \delta(q_k, M(e_{c,pub}))$. Therefore, if a robot chooses a public controllable event, all its peers are notified.

A. Free Behaviour Models

Figure 3 shows the additional free behaviour models that a robot uses when acting in a group. Free behaviour model G_3 represents the ability of the robot to request a particular basic movement to be performed: sF, sL, and sR. Free behaviour model G_4 represents the ability of the robot to receive the requests made by any other robot; the request (sF, sL, or sR) is provided via a public uncontrollable event ($\_sF$, $\_sL$, or $\_sR$). Free behaviour model G_5 represents a timer. Controllable event $startTimer$ activates the timer; and after 10 s, uncontrollable event $timeout$ occurs. Table I summarises all events; controllable, uncontrollable, public controllable, and public uncontrollable events are indicated as *C*, *U*, *PubC*, and *PubU*, respectively.



(a) G_3 (b) G_4 (c) G_5

Fig. 3. Additional free behaviour models for synchronous movement, representing the ability to choose a movement (a), receive a choice made by another robot (b), and a timer (c). Public controllable and public uncontrollable events are shown in blue and red, respectively.

TABLE I
EVENTS USED IN THE SYNCHRONOUS MOVEMENT CASE STUDY.

Event	Type	Description
mF, tL, tR	*C*	Move forward/left/right
rL, rR	*C*	Rotate left/right
oN	*U*	No object detected
oL, oR	*U*	Object detected on the left/right
sF, sL, sR	*PubC*	Request moving forwards/turning left/turning right
$\_sF$, $\_sL$, $\_sR$	*PubU*	Request received to move forwards/turning left/turning right
$startTimer$	*C*	Activate timer
$timeout$	*U*	10 s elapsed since timer activation

B. Control Specifications

Figure 4 shows the additional control specifications for acting in groups. Specifications E_2, E_3, and E_4 allow a robot to perform a basic movement only when in agreement with the last requests (if any) received by other robots. The movement is selected either by the robot itself (free behaviour model G_3), or in response to a request received by another robot (free behaviour model G_4). Each of E_2, E_3, and E_4 is concerned with a particular movement. For example, when a robot requests to move forward (i.e. event sF), the states of specifications E_2, E_3, and E_4 evolve to q_2, q_1, and q_1, respectively, which only allow the move forward event, mF, to be triggered. Note that an event is enabled within a robot only when enabled in all of the robot's generators that have that event in their alphabet. Events sF, sL, and sR are disabled by E_5 until a timeout expires. As public controllable event sF is mapped to public uncontrollable event $\_sF$, all other robots obtain the same state configuration, causing the group to perform a coherent movement. Specification E_5 defines a minimum time period between subsequent requests of 10 s.

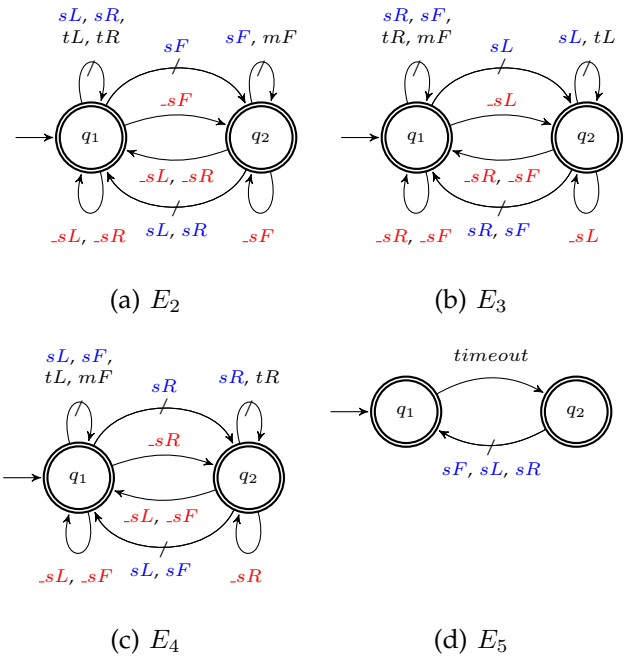


Fig. 4. Additional control specifications for the synchronous movement case study. Public controllable and public uncontrollable events are shown in blue and red, respectively.

C. Controller Synthesis

The supervisor represents the control logic of the robot. The supervisor is obtained by combining the free behaviour models and the specifications. This guarantees that the robot performs only actions that it can perform and should perform. Formally, this is achieved using synchronous composition, represented by $||$, (see [15]).

First, we join all free behaviour models and specifications into a target language, K:

$$G = G_1||\dots||G_5, \tag{3}$$
$$E = E_1||\dots||E_5, \tag{4}$$
$$K = G||E. \tag{5}$$

As K is not necessarily controllable [4], we extract the largest sub-language, S, that is controllable (see [4], [15]):

$$S = SupC(G, K), \tag{6}$$

S is a single generator (i.e. monolithic supervisor), which may be prohibitively large due to the combinatorial explosion of states during synchronous composition [16]. Two methods have been proposed to obtain modular supervisors that may have a reduced number of states. First, the modular supervisors [16] exploit the modularity of specifications. One target language and one supervisor are obtained for each specification, as:

$$K_i = G||E_i \quad \forall i \in \{1,\dots,5\}, \tag{7}$$
$$S_i = SupC(G, K_i) \quad \forall i \in \{1,\dots,5\}. \tag{8}$$

TABLE II
EVENTS USED IN EACH GENERATOR.

		E_1	E_2	E_3	E_4	E_5
G_1	mF	✓	✓	✓	✓	
	tL	✓	✓	✓	✓	
	tR	✓	✓	✓	✓	
	rL	✓				
	rR	✓				
G_2	oN	✓				
	oL	✓				
	oR	✓				
G_3	sF		✓	✓	✓	✓
	sL		✓	✓	✓	✓
	sR		✓	✓	✓	✓
G_4	$\_sF$		✓	✓	✓	
	$\_sL$		✓	✓	✓	
	$\_sR$		✓	✓	✓	
G_5	$startTime$					
	$timeout$					✓
local models		G_1^{loc}	G_2^{loc}	G_3^{loc}	G_4^{loc}	G_5^{loc}

Second, local modular supervisors [17] exploit in addition the modularity of the free behaviour models. In the synthesis of the local modular supervisor for each specification, only the free behaviour models that define the events for that specification are used.

For our case study, the relations are expressed in Table II. From this, we obtain the following local free behaviour models:

$$\begin{array}{lcl} G_1^{loc} & = & G_1||G_2, \\ G_2^{loc} = G_3^{loc} = G_4^{loc} & = & G_1||G_3||G_4, \\ G_5^{loc} & = & G_3||G_5. \end{array} \tag{9}$$

The target language and local modular supervisors for each specification are obtained by:

$$K_i^{loc} = G_i^{loc}||E_i \quad \forall i \in \{1,\dots,5\}. \tag{10}$$
$$S_i^{loc} = SupC(G_i^{loc}, K_i^{loc}) \quad \forall i \in \{1,\dots,5\}. \tag{11}$$

The monolithic supervisor, S, has 44 states and 420 transitions. The modular supervisors, S_i, have a total of 22 states and 286 transitions. The local modular supervisors, S_i^{loc}, have in total 13 states and 93 transitions.

V. IMPLEMENTATION

The framework implementation contains the generator player (i.e. a virtual machine to execute supervisors), the operational procedures (i.e. the interface to the hardware), and the communication stack.

We extended the open source software tool Nadzoru [18], [19] to support public events in the modelling of free behaviour models and specifications. Using the graphical interface, the user can flag an event as public. If the event is controllable, they also need to choose a corresponding public uncontrollable event. No further action is needed by the user. The implementation of the extended framework automatically generates the controller's source code.

Algorithm 1 Generator player (for N generators)

```
1:  procedure GENERATOR PLAYER
2:      for all j ∈ {1, 2, ..., N} do
3:          set current state c_j to initial state q_{0_j};
4:      end for
5:      while true do
6:          if e_u ∈ Σ_{u,pub} occurred externally then
7:              for all j ∈ {1, 2, ..., N} do
8:                  c_j = δ_j(c_j, e_u);
9:              end for
10:         else if e_u ∈ Σ_{u,pub} occurred internally then
11:             for all j ∈ {1, 2, ..., N} do
12:                 c_j = δ_j(c_j, e_u);
13:             end for
14:             Transmit( e_u );
15:         else if e_u ∈ Σ_{u,priv} occurred then
16:             for all j ∈ {1, 2, ..., N} do
17:                 c_j = δ_j(c_j, e_u);
18:             end for
19:         else
20:             Σ_c^enabled = subset of enabled events(Σ_c);
21:             if Σ_c^enabled ≠ ∅ then
22:                 select one e_c ∈ Σ_c^enabled;
23:                 for all j ∈ {1, 2, ..., N} do
24:                     c_j = δ_j(c_j, e_c);
25:                 end for
26:                 execute callback function of e_c;
27:                 if e_c ∈ Σ_{c,pub} then
28:                     Transmit( M(e_c) );
29:                 end if
30:             end if
31:         end if
32:     end while
33: end procedure
```

A. Generator Player

The generator player is shown in Algorithm 1. It is a modified version from the literature [20], [8]. It evolves the states of all generators of its robot. First, it checks if externally-created public (line 6), internally-created public (line 10), or private (line 15) uncontrollable events have occurred. For any uncontrollable event, the generators' states are updated accordingly (lines 7–9, 11–13, and 16–18). If a public uncontrollable event originated locally, then the group is notified (line 14).

If no uncontrollable event occurred, the generator player compiles a list of enabled (i.e. permissible) controllable events (line 20). Three situations can arise. First, the list is empty, indicating that no action can be triggered; the system stalls until an uncontrollable event happens. Second, the list contains only a single controllable event, which is the only available option. Third, the list contains multiple controllable events. In this case, all events are valid options that fulfill the control specifications; however, only one event can be triggered, this is known as the choice problem [21]. The generator player uniformly randomly selects one event and updates the generators' states accordingly (lines 23–25).

If a controllable public event occurred, then the swarm is notified (line 28) with the respective public uncontrollable event that is determined by function M.

Note that the generator player (controller) prioritises uncontrollable events over controllable events. This is important as the control decision must be taken based on the most recent state that is available. Moreover, the generator player prioritises public uncontrollable events over private uncontrollable events as public uncontrollable events affect the entire system.

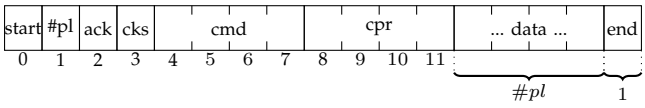


Fig. 5. Structure of packages to transmit public events.

B. Operational Procedures

The operational procedures interface the controller to the hardware [17]. They define one callback function per event and do not implement control logic. For each triggered controllable event, the generator player calls the respected callback function to perform the desired action (see line 26 of Algorithm 1). Moreover, the operational procedures implement the routines to determine whether an uncontrollable event has occurred locally (see lines 10 and 15).

C. Communication

The generator player operates on top of the underlying network implementation. In other words, the same supervisor can be deployed over different networks.

In the following experiment, we assume that the robots are connected to a central hub. The hub serves as a message delivery agent that transmits any incoming package to all other robots connected to it.

To establish a wireless network, we use Bluetooth, IEEE 802.15.1 [22], which is a package-based primary-secondary protocol. A single primary can communicate with up to seven secondaries.

We implemented communication by transmitting packages. The package structure is shown in Figure 5. Each package is composed of a header and data of a variable length. The header contains (1) the start byte (start), (2) the size in bytes of the data being transmitted (#pl), (3) an acknowledgement identification (ack), (4) the checksum value (cks), (5) the command (cmd), and (6) the command parameters (cpr). This is followed by (7) the data and (8) the end byte.

Whenever the hub receives a package from a robot, it reads the header and evaluates the checksum. If the package is valid, the hub emits an acknowledgement

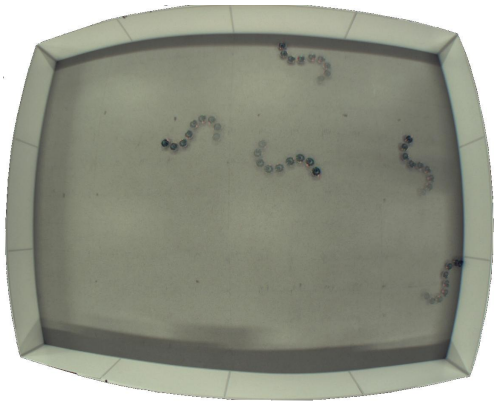


Fig. 6. Sequence of nine (superimposed) snapshots taken during a period of 3.2 s from one of the ten trials in which five e-pucks performed synchronous movements while avoiding collisions.

to the robot and performs the required action defined in the header's command element. Currently, the only commands implemented are: (1) *broadcasting*, where all robots receive a copy of the package and (2) *acknowledgment*, where the receiver indicates that the package was successfully received.

To perform the broadcast, the hub first queues one copy of the message to be transmitted in a local buffer, one per recipient robot. The message is then transmitted to the respective robot. To account for transmission errors the hub waits for an acknowledgement message. If the acknowledgement is not received within T^{ack} another attempt is made with an increased T^{ack}. After seven failed attempts, the message is discarded. If the acknowledgement is received the message is removed from the buffer. Robot and hub use the same acknowledgement process.

VI. EXPERIMENTS

To validate the implementation of the SCT extension in practice, we performed experiments using the mobile robotics platform e-puck [23]. Each robot uses a copy of the same set of local modular supervisors.

Trials were performed in a $400\,\mathrm{cm} \times 300\,\mathrm{cm}$ light grey floored arena surrounded by white walls that were $50\,\mathrm{cm}$ in height. The arena had 165 pencil marks distributed as a 15×11 grid with columns and rows spaced $25\,\mathrm{cm}$ apart. Five e-pucks were uniformly randomly distributed over the marks with a randomly uniformly distributed orientation. Ten trials were performed, each lasting $300\,\mathrm{s}$. With a remote control, the robots were instructed to start simultaneously.

Figure 6 shows snapshots taken from one of the experimental trials in which the e-pucks perform synchronous movements while avoiding collisions with each other and the walls of the arena. The nine snapshots are superimposed to show how the robots are synchronised, the first snapshot is the most transparent one and the last the most opaque.

Video recordings from all 10 experimental trials and additional resources (models, the Nadzoru tool, the used source code) can be found in the online supplementary material [24].

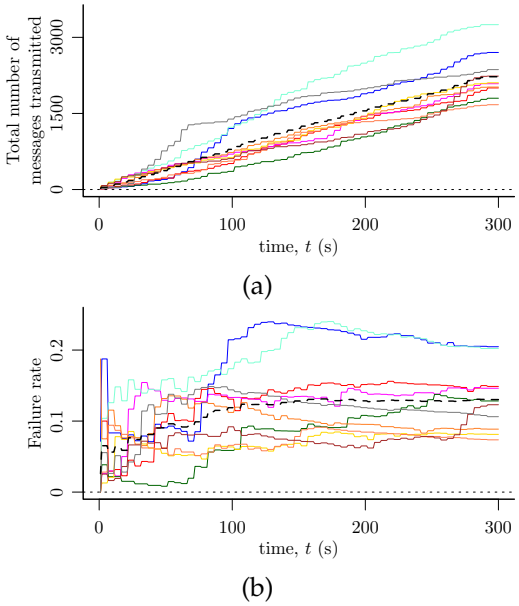


Fig. 7. Communication reliability: (a) Number of message transmission attempts by the hub to the robots. (b) Failure rate. The thick black dashed line indicates the mean, each other colour is a trial.

To evaluate the performance of the communication, the hub collected statistics about the transmission of packages. The transmission of every occurrence of a public event is attempted up to seven times, which was sufficient to successfully transmit packages during the experiments. Figure 7(a) shows the accumulated number of message transmission attempts by the hub to the robots. Figure 7(b) shows the failure rate of the messages against the total of attempted messages for each of the five robots, as well as the mean (dashed line).

VII. CONCLUSIONS

In this paper, we introduced the concept of public events for the supervisory control of swarms of robots. This makes it possible to formalise not only how robots act individually within a group, but also how they interact, as the robots exchange information. The user can define desired robot inter-dependencies at the highest level—the control specifications. Public events can be observed by other robots of the swarm, abstracting the communication between them. They are modelled as uncontrollable events. If a public controllable event must be transmitted, a function converts it into a corresponding public uncontrollable event.

To illustrate the proposed framework, a case study was presented. Experiments with 5 e-puck robots showed that the robots could synchronise their movements in space and time, while avoiding collisions with obstacles. We assumed that all robots can communicate with each other at all times. In the future, we will explore more complex environments, where local communication restrictions apply.

REFERENCES

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[2] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event process," *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[3] ——, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

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

[5] W. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history 1980-2015," in *20th World Congress*, 2017.

[6] A. B. Leal, D. L. L. Cruz, and M. S. Hounsell, "Supervisory control implementation into programmable logic controllers," *14th IEEE International Conference on Emerging Technologies and Factory Automation - ETFA*, 2009.

[7] A. D. Vieira, E. A. P. Santos, M. H. de Queiroz, A. B. Leal, A. D. de Paula Neto, and J. E. R. Cury, "A method for plc implementation of supervisory control of discrete event systems," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 1, pp. 175–191, Jan 2017.

[8] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Supervisory control theory applied to swarm robotics," *Swarm Intelligence*, vol. 10, no. 1, pp. 65–97, 2016.

[9] R. C. Hill and S. Lafortune, "Scaling the formal synthesis of supervisory control software for multiple robot systems," in *2017 American Control Conference (ACC)*, May 2017, pp. 3840–3847.

[10] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Probabilistic supervisory control theory (pSCT) applied to swarm robotics," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '17. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 1395–1403.

[11] J. A. Dulce-Galindo, M. A. Santos, G. V. Raffo, and P. N. Pena, "Autonomous navigation of multiple robots using supervisory control theory," in *2019 18th European Control Conference (ECC)*, June 2019, pp. 3198–3203.

[12] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari, "Automode: A novel approach to the automatic design of control software for robot swarms," *Swarm Intelligence*, vol. 8, no. 2, pp. 89–112, 2014.

[13] N. Palomeras, P. Ridao, M. Carreras, and C. Silvestre, "Using petri nets to specify and execute missions for autonomous underwater vehicles," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, oct. 2009, pp. 4439–4444.

[14] D. Bruneo, M. Scarpa, A. Bobbio, D. Cerotti, and M. Gribaudo, "Markovian agent modeling swarm intelligence algorithms in wireless sensor networks," *Performance Evaluation*, vol. 69, no. 3-4, pp. 135–149, 2012.

[15] W. Wonham, "Supervisory control of discrete event systems," Dept. Elect. Comput. Eng, Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2010.

[16] W. Wonham and P. Ramadge, "Modular supervisory control of discrete event system," *Mathematics of control, signals and systems*, vol. 1, no. 1, pp. 13–30, 1988.

[17] M. Queiroz and J. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell," in *Proceedings of 6th International Workshop on Discrete Event Systems (WODES)*. Piscataway, NJ: IEEE, 2002, pp. 103–110.

[18] Y. K. Lopes, A. B. Leal, R. S. U. Rosso, and E. Harbs, "Local modular supervisory implementation in microcontroller," in *Proceedings of the 9th International Conference of Modeling, Optimization and Simulation (MOSIM 2012)*, 2012.

[19] L. P. Pinheiro, Y. K. Lopes, A. B. Leal, and R. S. U. Rosso, "Nadzoru: A software tool for supervisory control of discrete event systems," in *IFAC-PapersOnLine*, vol. 48, 2015, pp. 182–187.

[20] Y. K. Lopes, A. B. Leal, T. J. Dodd, and R. Groß, "Application of supervisory control theory to swarms of e-puck and kilobot robots," in *Swarm Intelligence, ANTS 2014*, ser. LNCS, M. Dorigo, et al., Ed., vol. 8667. Berlin, Germany: Springer, 2014, pp. 62–73.

[21] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *1998 IEEE 37th Conference on Decision and Control*, vol. 3. Piscataway, NJ: IEEE, 1998, pp. 3305–3310.

[22] *IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. - Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*. IEEE, 2005.

[23] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, no. 1, 2009, pp. 59–65.

[24] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Electronic Supplementary Material," 2020. [Online]. Available: http://naturalrobotics.group.shef.ac.uk/supp/2020-001/