

Supervisory control theory applied to swarm robotics

Yuri K. Lopes¹ · Stefan M. Trenkwalder¹ ·
André B. Leal² · Tony J. Dodd¹ · Roderich Groß¹

Received: 16 June 2015 / Accepted: 27 January 2016 / Published online: 25 February 2016
© The Author(s) 2016. This article is published with open access at Springerlink.com

Abstract Currently, the control software of swarm robotics systems is created by ad hoc development. This makes it hard to deploy these systems in real-world scenarios. In particular, it is difficult to maintain, analyse, or verify the systems. Formal methods can contribute to overcome these problems. However, they usually do not guarantee that the implementation matches the specification, because the system's control code is typically generated manually. Also, there is cultural resistance to apply formal methods; they may be perceived as an additional step that does not add value to the final product. To address these problems, we propose supervisory control theory for the domain of swarm robotics. The advantages of supervisory control theory, and its associated tools, are a reduction in the amount of ad hoc development, the automatic generation of control code from modelled specifications, proofs of properties over generated control code, and the reusability of formally designed controllers between different robotic platforms. These advantages are demonstrated in four case studies using the e-puck and Kilobot robot platforms. Experiments with up to 600 physical robots

Electronic supplementary material The online version of this article (doi:[10.1007/s11721-016-0119-0](https://doi.org/10.1007/s11721-016-0119-0)) contains supplementary material, which is available to authorized users.

✉ Roderich Groß
r.gross@sheffield.ac.uk

Yuri K. Lopes
y.kaszubowski@sheffield.ac.uk

Stefan M. Trenkwalder
s.trenkwalder@sheffield.ac.uk

André B. Leal
andre.leal@udesc.br

Tony J. Dodd
t.j.dodd@sheffield.ac.uk

¹ Department of Automatic Control and Systems Engineering, The University of Sheffield, Mappin Street, Sheffield S1 3JD, UK

² Department of Electrical Engineering, Santa Catarina State University, Campus Universitário Prof. Avelino Marcante, Joinville, SC 89219-710, Brazil

are reported, which show that supervisory control theory can be used to formally develop state-of-the-art solutions to a range of problems in swarm robotics.

Keywords Supervisory control theory · Swarm robotics · Formal methods · Kilobot · e-puck · Automatic code generation

1 Introduction

Swarm robotics studies how large groups of robots can interact with each other in simple ways to solve relatively complex tasks cooperatively. Swarm robotics systems may accomplish tasks despite failures in some of the robots, and they are typically designed so that their performance scales well with the number of robots. These properties can be useful in several applications (Brambilla et al. 2013).

Designing the control logic for a swarm of robots is a challenging problem. Each robot in the swarm typically executes an identical program which has access only to a limited amount of local information. As a consequence, it is unaware of the overall configuration of the swarm.

The control software of swarm robotics systems is usually obtained through ad hoc development, without relying on software engineering methods. The ad hoc development, which is mainly used in academic environments, hinders the transition of swarm robotics systems to real-world applications. The source code resulting from ad hoc development is difficult to be maintained, analysed, or verified.

Formal methods help in addressing these problems (for example, see Knight et al. (1997)). They require a systematic formalisation of the solutions. Such formalisation can be subjected to analysis tools, for example, to verify that certain properties are met. They also serve as a documentation of the system.

However, even when formal methods were used, it was not guaranteed that the final source code would accurately represent the specifications. This is because the source code was obtained in a manual process as automatic code generation has not been supported yet.

Furthermore, a cultural resistance to apply formal methods can be observed: they may be perceived as an additional step that does not add value to the final product, prolongs its development cycle, and introduces undesired complexity, and their integration is often impeded by the lack of appropriate tools (Knight et al. 1997).

In this paper¹, we propose the application of supervisory control theory (SCT) to the domain of swarm robotics. SCT is a framework for formally synthesising controllers, also referred to as supervisors (Ramadge and Wonham 1987). In SCT, formal languages are used to model the capabilities of systems. At the same time, specifications, also expressed as formal languages, are used to restrict these capabilities. This ensures that the system behaves as intended. By using SCT, we show how to formally model the capabilities of and specifications for swarm robotics systems and how to automatically generate the controllers (including source code) for the individual robots in the swarm.

The contributions of this work are: (1) the proposal and demonstration of the use of SCT for formally developing controllers in swarm robotics; (2) the application of SCT using the full Ramadge and Wonham (RW) framework (Ramadge and Wonham 1987), from the mod-

¹ This paper is an extension of (Lopes et al. 2014). It presents three new case studies (Sects. 4.2, 4.3, 4.4) and related experiments (Sects. 7.2, 7.3, 7.4), an analysis of three methods for synthesising formal controllers (Sects. 5.1, 5.2, 5.3, 5.4), and a more comprehensive description of the implementation of the synthesised controllers (Sect. 6).

elling to the software implementation; (3) the adaptation of an open source software tool to automatically generate the control software for a swarm of robots; and (4) the comparison of three existing control synthesis methods, monolithic (Ramadge and Wonham 1987), modular (Wonham and Ramadge 1988), and local modular (Queiroz and Cury 2000b, a, 2002) in four swarm robotics' case studies.

Two case studies illustrate how the canonical problems of *aggregation* (Gauci et al. 2014a) and *object clustering* (Gauci et al. 2014b) can be formalised using the SCT framework. Two further case studies report novel solutions to *segregation* and *group formation* problems. These case studies are more advanced and explore more features of SCT. The results from the case studies clearly demonstrate the potential of using SCT in swarm robotics.

This paper is organised as follows: Sect. 2 reviews related works. Section 3 introduces SCT. Section 4 presents the case studies and how they are modelled in the SCT framework. Section 5 shows the control synthesis using SCT. Section 6 details the implementation. Section 7 presents the experiments. Finally, Sect. 8 concludes the paper.

2 Formal methods in swarm robotics

This section overviews previous work that investigates the application of formal methods in swarm robotics. Swarm robotics systems can be modelled at multiple levels of abstraction. Microscopic models capture the behaviours of individual members of swarm systems, while macroscopic models capture the behaviours of the systems as a whole. In Martinoli et al. (2004), the authors present a unified framework for modelling a swarm robotics system at both microscopic and macroscopic levels. The framework uses probabilistic finite state machines (PFSM). The authors present a case study where a swarm of robots cooperatively pull sticks out of the ground.

As Massink et al. (2013) argue, modelling a system at multiple levels (microscopic and macroscopic) can lead to inconsistencies. To address this, they propose biochemical performance evaluation process algebra (Bio-PEPA), a method widely used for modelling biochemical reactions. Bio-PEPA models the swarm robotics system solely at the microscopic level and supports the integration of spatial information. Bio-PEPA enables analysis and verification of macroscopic features by model checking (Massink et al. 2013).

In the work of Tanner et al. (2007), algebraic graph theory is applied to analyse stability properties of networked mobile agents that are flocking using decentralised control. Agents exchange information via networks that may change in topology. They do so while avoiding collisions and converging to a common direction and speed.

Brambilla et al. (2015) propose a method called property-driven design. The method consists of four steps: first, the requirements are formally specified; second, a prescriptive macroscopic model is designed using Markov chains and verified by model checking; third, this model is used to guide the implementation of a simulated robot swarm; and finally, the desired robot swarm is implemented and tested in simulation and then with robots. Property-driven design does not yet incorporate the automatic porting of the models to source code.

Probabilistic finite state machines can be automatically generated. In Francesca et al. (2014b), the AutoMoDe-Vanilla (automatic modular design) approach is presented. In AutoMoDe predefined parametric modules serve as building blocks to the design process. Controllers, represented as PFSM, are obtained using the F-Race optimisation algorithm. They are then compared against controllers generated by EvoStick (Francesca et al. 2014b) and by human experts (Francesca et al. 2014a). The human experts are either constrained to

use the predefined parametric modules (C-Human) or unconstrained (U-Human). The results show that C-Human outperformed AutoMoDe-Vanilla, but AutoMoDe-Vanilla was better than U-Human. AutoMoDe was also extended to use the iterated F-Race algorithm, resulting in an approach called AutoMoDe-Chocolate, which outperforms both AutoMoDe-Vanilla and C-Human (Francesca et al. 2015).

Petri-nets are another formal approach to modelling the control software of swarm robotics systems. In King et al. (2003), the authors use Petri-nets to coordinate the actions of a group of robots. The Petri-net-based controller is, however, executed from a central computer, which communicates with the robots. Petri-nets for multi-robot control have been analysed to detect properties such as boundedness, livelocks, and deadlocks (Costelha and Lima 2008).

Temporal logic (Emerson 1990) has been applied to model and analyse swarm robotics systems (Winfield et al. 2005). It can be combined with model checking for formal verification (Belta et al. 2007; Dixon et al. 2011, 2012). In model checking, all possible executions of the system are considered to check whether certain properties are met. Automatic code generation for temporal logic models has yet to be shown.

SCT (Ramadge and Wonham 1987, 1989) provides an alternative approach to formally developing controllers. SCT is mostly applied in the context of manufacturing systems. It is used to synthesise a controller based on formal descriptions of the system and specifications. Studies have illustrated how code for manufacturing coordination control can be automatically generated using SCT (Liu and Darabi 2002; Queiroz and Cury 2002; Lopes et al. 2012). SCT is also applied to design controllers for systems of multiple robots (Gordon-Spears and Kiriakidis 2004; Tsalatsanis et al. 2009, 2012), solving tasks such as object delivery and patrolling/inspection. These works focus, however, on the design and analysis of controllers rather than on their implementation and/or validation using physical robots. Moreover, the works by Tsalatsanis et al. (2009, 2012) are only partially based on the RW framework (Ramadge and Wonham 1987, 1989); as a consequence, a variety of software tools and theory are not applicable to those systems. SCT has also been considered for transportation systems, moving both goods (Silva et al. 2008; Mass et al. 2012) and persons, for example, in theme parks (Forschelen et al. 2012).

SCT, as most of the aforementioned methods, assumes discrete system states. Hybrid system theory (HST) offers an alternative, where the system can be represented by both discrete and continuous states. HST has been used in the context of swarm robotics, multi-robot systems, and other multi-agent systems (Tomlin et al. 1998; Fierro et al. 2001; McNew and Klavins 2006; McNew et al. 2007; Zavlanos et al. 2009; Mesquita 2010; Mesquita and Hespanha 2012).

Belta et al. (2007), for example, use HST in a motion planning problem. The state of the robot reflects its position in the environment. It is represented both discretely and continuously. For the discrete representation, the environment is partitioned into a graph of triangular regions. The graph can be used to express high-level specifications using temporal logic. The paths in the graph that conform with the specification can be considered as words of a formal language. These can be recognised by a discrete automaton. The continuous representation is used to realise low-level motion control, for example, to move the robot from one triangular region to the next. The overall controller can thus be considered a hybrid automaton.

Fierro et al. (2001) use HST to control the formation of a group of three robots moving along a given trajectory. Here the discretisation is at the controller level: the robot has multiple continuous motion controllers. Its sensory input—which other robots are perceived—is used to select the controller to be executed. In McNew and Klavins (2006); McNew et al. (2007), HST is used for the problem of organising the robots into subgroups while maintaining the

overall connectivity. The formal properties are guaranteed by the use of embedded graph grammars.

As shown in this paper, SCT allows for automatic code generation for a swarm of robots. While formal methods have been used for the design and validation of swarm robotics control before, they do not guarantee that the controller implementation faithfully represents the specification. We address this issue by generating the control software automatically using SCT. The control software is formalised using regular languages. Deterministic finite automata, which realise these languages, are already widely used in the control of swarm robotics systems.

3 Supervisory control theory preliminaries

SCT (Ramadge and Wonham 1987; Wonham and Ramadge 1988; Ramadge and Wonham 1989) is a theoretical framework for synthesising controllers, called supervisors. It assumes that the systems under investigation can be represented as discrete event systems (DES).² DES are composed of discrete states. Changes in state (called transitions) are triggered by events (Cassandras and Lafortune 2008). SCT distinguishes between uncontrollable events and controllable events. Uncontrollable events represent feedback signals, for example, from sensors. Controllable events represent command signals—issued by the controller, for example, to move a robot forward.

In SCT, the designer models (i) what the system can do and (ii) what it should do. Concerning (i), they specify an arbitrary number of so-called *free behaviour models*, which describe all of the system’s capabilities. Concerning (ii), they specify an arbitrary number of so-called *control specifications*. Both free behaviour models and control specifications are expressed using a formal language. A language is a set of words that are composed of symbols over an alphabet. Each symbol corresponds to an event of the DES. Therefore, the desirable sequence of events form the words of the language. SCT combines all free behaviour models and control specifications into a coherent language. It synthesises a supervisor (controller), which guarantees that, at any time, only valid words or prefixes of valid words occur. This is realised by restricting the set of controllable events that the system may choose from. For example, consider a service robot tasked to retrieve milk from a fridge. The robot would first choose controllable event “open fridge”. Suppose uncontrollable event “fridge has milk” was then triggered; SCT would restrict the set of controllable events to “take milk out of fridge” and “close fridge” thereafter. If, however, uncontrollable event “fridge out of milk” was triggered, SCT would restrict the set of controllable events to “close fridge”. In both cases the robot would be prevented from starting a new activity until the fridge was closed. The desired sequence of events—related to actuation and sensing—would thus adhere to what is both possible and desirable, either (“open fridge”, “fridge out of milk”, “close fridge”) or (“open fridge”, “fridge has milk”, “take milk out of fridge”, “close fridge”).

3.1 Generators

The class of formal languages that is most commonly used in SCT are the regular languages, also called Type-3 languages (Chomsky 1956, 1959). Words of a regular language, within the SCT framework, can be produced by a generator. A generator is similar to a finite automaton, also called finite state machine (FSM). However, while a finite automaton recognises words

² When hybridised with continuous methods, however, SCT can also be applied to continuous systems.

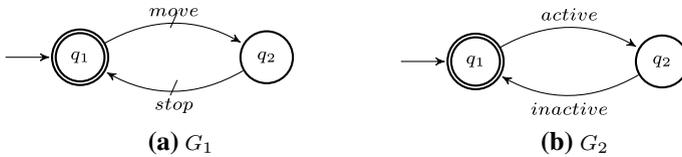


Fig. 1 Example of free behaviour models for **a** a conveyor and **b** a sensor placed at the end of the conveyor. Each behaviour model G_i has its own set of states $Q_i = \{q_1, q_2, \dots\}$. *move* and *stop* are controllable events. *active* and *inactive* are uncontrollable events

from a particular regular language (i.e. given a word the automaton will accept it or not accept it), a generator produces words that belong to the language. A generator G is a 5-tuple $(Q, \Sigma, \delta, q_0, Q_m)$, where Q is a finite set of states; Σ is a finite set of symbols related to the system's events; $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function; $q_0 \in Q$ is the initial state; and $Q_m \subseteq Q$ is a set of marked states. The language realised by generator G is referred to as $L(G)$. For simplicity, we may use G indistinctly to denote the generator or the language $L(G)$.

Events (that are the symbols of the language) are of two types: uncontrollable events (Σ_u) and controllable events (Σ_c), where $\Sigma = \Sigma_u \cup \Sigma_c$ and $\Sigma_u \cap \Sigma_c = \emptyset$. A controllable event $e_c \in \Sigma_c$ is enabled in a state $q \in Q$ if $\delta(q, e_c)$ is defined. Let Σ^* denote the set of all words—or sequences of events—over an alphabet Σ . Let Σ^+ denote the set of all words excluding the empty word ϵ (i.e. $\Sigma^+ = \Sigma^* \setminus \epsilon$).

Marked states are states that are considered safe for the system. For example, a marked state can correspond to the end of a task. Reaching a marked state does not necessarily implicate the end of the operation; the generator could continue to evolve.

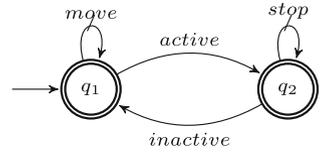
3.2 Free behaviour models

In SCT, the system is formally represented by m free behaviour models. Each free behaviour model abstracts one of the system's relevant physical capabilities. This modularisation leads to an intuitive link between hardware and software (also, see [Cowley and Taylor \(2007\)](#)). The free behaviour modules are realised by generators G_i , $i \in \{1, 2, \dots, m\}$. By default, it is assumed that the free behaviour models are independent of each other. Figure 1 shows two examples of free behaviour models. These represent (a) a conveyor that transports parts in a manufacturing plant and (b) a sensor that detects the presence of a part at the end of the conveyor. States are represented by circles. The initial state is indicated by an unlabelled arrow. Marked states are represented by double-line circles. It is common that only the initial state of a free behaviour model is marked. This means that the resulting supervisors should be able to return to the initial condition. Transitions and associated events are shown as labelled arrows. Arrows with a stroke relate to controllable events, and arrows without a stroke relate to uncontrollable events.

3.3 Control specifications

The desired behaviour of the system is formally represented by n control specifications. Each control specification restricts the possibilities of one or more free behaviour models. It is realised by a generator E_j , $j \in \{1, 2, \dots, n\}$. Figure 2 shows an example specification that relates the free behaviour model of the conveyor with that of the sensor to implement the following rule: “when a part arrives in front of the sensor the conveyor shall stop, otherwise

Fig. 2 Example of a control specification that enables the conveyor to move only when there is no part in front of the sensor



it shall move”. SCT works by preventing controllable events from occurring in some states. This is achieved by disabling controllable events. For example, in state q_1 (see Fig. 2), event *stop* is disabled and event *move* is enabled. Hence, when the sensor is inactive, the conveyor will move. Normally, all states of specifications are marked states. An exception to this would be a specification representing a buffer. It can then be desirable, for a system, to guarantee that it reaches a state with the buffer empty; thus, only the state that represents the empty buffer is marked.

4 Design of free behaviour models and control specifications

SCT models the system and its specifications using formal languages. The modelling process may not always be intuitive, and multiple models may represent the same system or specification. In the following, we provide guidance on how to model systems with SCT. We present four case studies that illustrate how SCT can be applied in swarm robotics. The case studies make use of two robotic platforms, the Kilobot (Rubenstein et al. 2012) and the e-puck (Mondada et al. 2009). Both platforms move on the ground and are able to locally broadcast messages.³ The Kilobot is equipped with a light sensor and an RGB LED. The e-puck is equipped with an on-board camera and several LEDs distributed along its perimeter.

One case study—*segregation*—uses both robotic platforms. It shows that as long as all task-relevant hardware is available, the same supervisors can be applied to different robotic platforms. Two further case studies, using the e-puck platform, illustrate how state-of-the-art solutions for the problems of *aggregation* (Gauci et al. 2014a) and *object clustering* (Gauci et al. 2014b) can be formalised using the SCT framework. The last case study—*group formation*—requires advanced features of SCT. It uses the Kilobot platform.

We use $\theta \in \{s, a, c, g\}$ to refer to the different case studies, where *s* refers to segregation, *a* to aggregation, *c* to object clustering, and *g* to group formation. G_i^θ denotes the *i*th of m^θ free behaviour models and E_j^θ denotes the *j*th of n^θ control specifications.

4.1 Segregation case study

The system comprises an arbitrary number of leader and follower robots. Each leader assumes one of multiple types, characterised by its colour. Here, colours red, green, and blue are assigned at the beginning of the experiment. The segregation strategy separates follower robots into distinct groups, whereby each follower robot belongs to at most one leader (Lopes et al. 2014). Each leader broadcasts a signal containing its colour within a limited range. Follower robots within the signal range of only one type of leader belong to that leader and do not move. Followers that do not receive a signal also do not move. Followers that receive a signal from more than one type of leader move randomly.

The free behaviour models are illustrated in Fig. 3. G_1^s represents a user input device that triggers the uncontrollable event *press*. It is used to configure the robot. G_2^s defines the robot

³ The e-puck requires a non-standard library to broadcast messages using infrared.

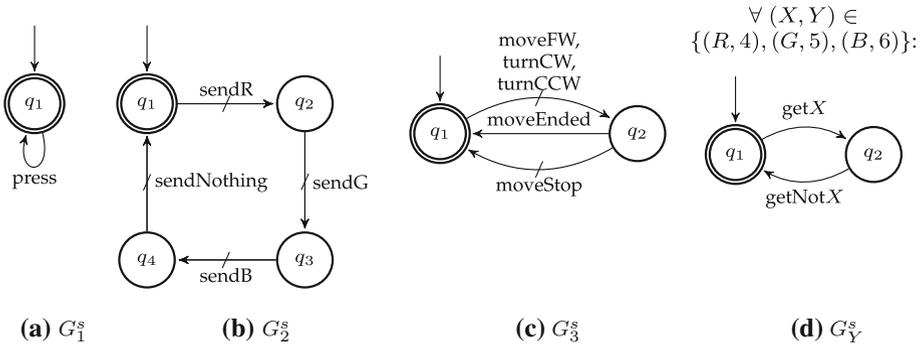


Fig. 3 Free behaviour models for the segregation case study. **a** Input device to configure the robot; **b** the robot’s ability to assume one of the three leader types or to be a follower; **c** motion capabilities; **d** the robot’s ability to receive messages from nearby leader robots

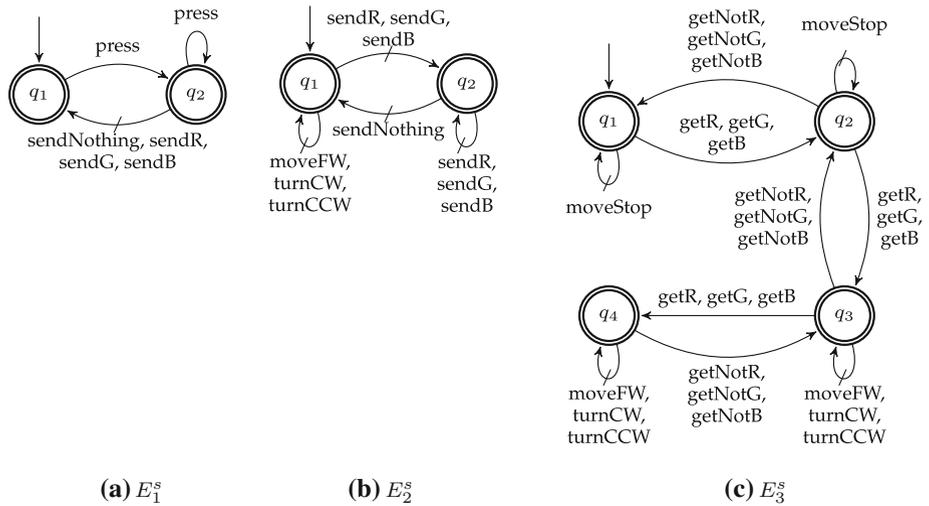


Fig. 4 Specification for the segregation case study. **a** Configures the robot through user interaction; **b** allows followers to move and leaders to transmit a signal; **c** moves the follower robot according to the signal received

type. By default, the robot is a follower (state q_1 in G_2^s). Followers do not transmit any message. There are three types of leaders, which are set by the controllable events $sendR$, $sendG$, and $sendB$. These events enable the broadcast of the messages red, green, and blue, respectively. The follower type can be set by the controllable event $sendNothing$. G_3^s represents the robot’s motion capabilities. The motion is started through controllable events $moveFW$ (move forward), $turnCW$ (turn clockwise), and $turnCCW$ (turn counter clockwise). The motion proceeds for a random period of time, and then, the uncontrollable event $moveEnded$ is generated. The motion can also be stopped by the controller through controllable event $moveStop$.

G_4^s , G_5^s , and G_6^s represent three different sensor outcomes that detect the presence of red, green, and blue leaders, respectively. The corresponding uncontrollable events $getX$, $X \in \{R, G, B\}$ indicate that the robot has received a message, respectively, from a red, green, or blue leader during the sample period of 0.2 s. On the other hand, the event $getNotX$, $X \in \{R, G, B\}$ occurs if no such message was received.

Fig. 5 Free behaviour models for the aggregation case study. **a** Binary sensor; **b** motion capabilities

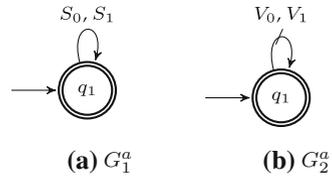


Figure 4 shows the specifications for the segregation strategy. The user can configure the robot type. When *press* occurs, specification model E_1^s reaches state q_2 , where the events *sendR*, *sendG*, *sendB*, and *sendNothing* are enabled. As seen in model G_2^s (Fig. 3b), the robot type is restricted to change sequentially through states of follower (q_1), red leader (q_2), green leader (q_3), and blue leader (q_4). Specification E_2^s allows only followers to move. It also sets up the message broadcasting of leaders. The main strategy is represented in specification E_3^s , where only the stop event (*moveStop*) is enabled while being in state q_1 (no signal received) or state q_2 (one type of leader signal received). Consequently, the robot does not move. However, if signals of two types of leaders (state q_3) or all three types of leaders (state q_4) are received, the previously described motion events are enabled. The controller can, therefore, choose from all three movement options. How this choice is made is an implementation question (for details about the choice problem, see (Fabian and Hellgren 1998)). In this work, the options are chosen with equal probability. It is worth noting that message receiving events *getR* and *getNotR* occur alternatively (see free behaviour model G_4^s in Fig. 3d); the same is true for *getG/getNotG* as well as for *getB/getNotB*. This property is exploited in control specification E_3^s .

4.2 Aggregation

The aggregation strategy allows a group of e-puck robots to gather in a homogeneous environment (Gauci et al. 2014a). It requires each robot to be equipped with a binary sensor, I , which detects the presence of other robots in its line of sight. The sensor provides a value $I = 1$ if there is a robot in the line of sight and $I = 0$ otherwise. For this setting, Gauci et al. (2014a) propose a reactive controller: if no other robot was detected, the robot would move backward along a circular trajectory, with scaled wheel velocities $(v_{l0}, v_{r0}) = (-0.7, -1)$. If another robot was detected, the robot would turn clockwise on the spot, with scaled wheel velocities $(v_{l1}, v_{r1}) = (1, -1)$. This controller was shown to be provably correct for two robots and it performed the aggregation task reliably with 40 physical robots (Gauci et al. 2014a). In the following, we show how to formalise this controller using SCT.

Figure 5 shows the free behaviour models for the aggregation strategy. Free behaviour model G_1^a represents the binary sensor; uncontrollable events S_0 and S_1 represent sensor readings $I = 0$ and $I = 1$, respectively. Free behaviour model G_2^a represents the possible movements. Controllable events V_0 and V_1 represent the pairs (v_{l0}, v_{r0}) and (v_{l1}, v_{r1}) , respectively. The movements are executed until a new command is issued.

Figure 6 shows the specifications to implement the aggregation strategy. Specification E_1^a enables event V_0 (robot moving backward along circular trajectory) if no other robot is perceived (S_0). Specification E_2^a enables event V_1 (robot turning on the spot) if another robot is perceived (S_1). Specifications E_3^a and E_4^a guarantee that events V_0 and V_1 will occur in alternation. Note that the movement of a robot continues indefinitely once V_0 or V_1 is triggered.

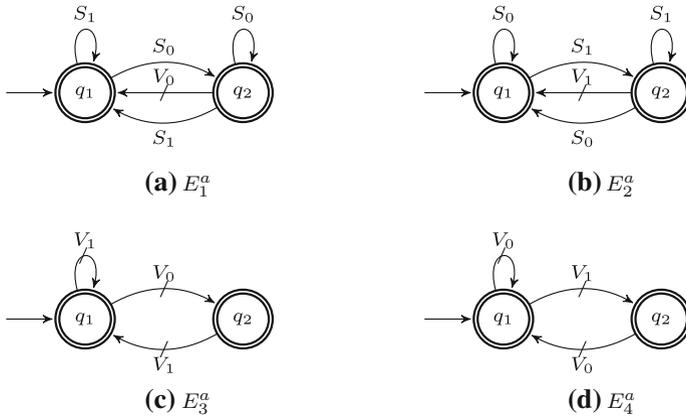


Fig. 6 Specification for the aggregation case study. **a, b** Move the robot according to the sensor reading; **c, d** prevent the same movement event from occurring consecutively (yet, the robot will perform its current movement indefinitely)

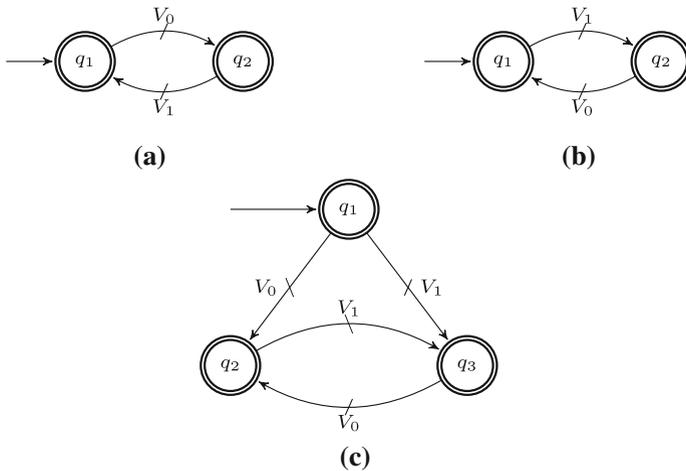


Fig. 7 Three alternative specifications to be used instead of E_3^a and E_4^a in the aggregation case study. The specifications in **a** or **b** are not fully equivalent to E_3^a and E_4^a , as they restrict the first movement to be either V_0 or V_1 , respectively. The specification in **c** is the result of synchronous composition $E_3^a || E_4^a$ and thus fully equivalent

Instead of specifications E_3^a and E_4^a , one may apply one of the specifications shown in Fig. 7. Note that the specification shown in Fig. 7a would not be fully equivalent to E_3^a and E_4^a , as it forces V_0 to occur first. Similarly, the specification shown in Fig. 7b forces V_1 to occur first. The third option, Fig. 7c, is, however, equivalent to E_3^a and E_4^a ; it can be obtained by their synchronous composition (for details, see Sect. 5).

4.3 Object clustering

The object clustering strategy allows a group of e-puck robots to cluster objects that are initially dispersed in the environment (Gauci et al. 2014b). Each robot can detect the presence

Fig. 8 Free behaviour models for the object clustering case study. **a** Sensor to detect nothing, objects, or robots; **b** motion capabilities

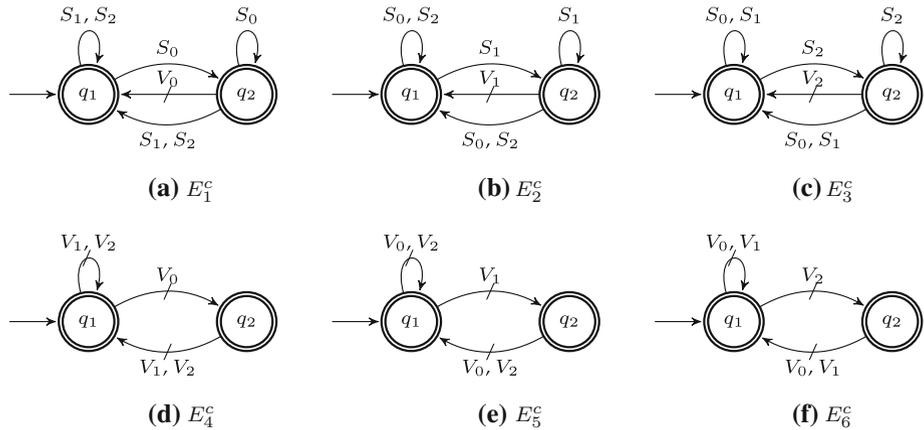
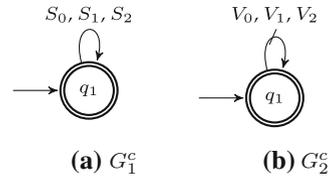


Fig. 9 Specification for the object clustering case study. **a–c** Move the robot according to the sensor reading; **d–f** prevent the same movement event from occurring consecutively (yet, the robot will perform its current movement indefinitely)

of an object or another robot in its direct line of sight. Its line-of-sight sensor I thus indicates what it is pointing at: $I = 0$ if it is pointing at nothing (or the walls of the environment, if this is bounded); $I = 1$ if it is pointing at an object; and $I = 2$ if it is pointing at another robot.

The free behaviour models for this strategy are similar to those presented in the aggregation case study. However, there are three modes of movement instead of two. The corresponding parameters were obtained using an evolutionary search (Gauci et al. 2014b):

$$x = (v_{l0}, v_{r0}, v_{l1}, v_{r1}, v_{l2}, v_{r2}) = (0.5, 1, 1, 0.5, 0, 0.5), \tag{1}$$

where, v_{lI} and v_{rI} are the left and right wheel velocities, respectively, when the sensor reading is I .

Figure 8 shows the free behaviour models for this strategy. Free behaviour G_1^c represents the sensor. Uncontrollable events S_0 , S_1 , and S_2 represent the presence of nothing ($I = 0$), an object ($I = 1$), or another robot ($I = 2$) in the line of sight. Free behaviour G_2^c defines the possible movements. Controllable events V_0 , V_1 , and V_2 represent pairs (v_{l0}, v_{r0}) , (v_{l1}, v_{r1}) , and (v_{l2}, v_{r2}) , respectively. The movements are executed indefinitely.

Figure 9 illustrates the specifications for the object clustering strategy. Specifications E_1^c , E_2^c , and E_3^c relate, respectively, the perception of nothing (S_0), an object (S_1), or another robot (S_2) with the wheel velocities, which are specified by parameters v_{lI} and v_{rI} through controllable events V_I , $I \in \{0, 1, 2\}$. Specifications E_4^c , E_5^c , and E_6^c guarantee that events V_0 , V_1 , and V_2 occur in alternation (e.g., when event V_0 occurs it cannot occur again until either event V_1 or event V_2 occurs).

4.4 Group formation

This case study is performed with the Kilobot robots. It involves two types of robots, leaders and followers. The followers are of two classes, which we call green and blue. Robots of the same class could be equipped with identical tools, for example. The task is to group each leader with a balanced number of followers from each class (± 1). We call this the *equilibrium criterion*.

The strategy is as follows. Leaders are randomly distributed over the arena and do not move. Followers move randomly broadcasting a message (**broadcast**) containing a unique identification code, their class, and the message type. When a leader receives a **broadcast** message, it sends an **offer** message if adding that robot would fulfil the equilibrium criterion. When a follower receives an **offer**, it stops, sends an **acceptance** message to the leader, and starts relaying any message to and from their leader.

Figure 10 shows the free behaviour models for this strategy. Free behaviour model G_1^g represents the movement capabilities. It is identical to G_3^g in Fig. 3c. Free behaviour model G_2^g represents a timer. Controllable event *startTimer* starts the timer. After the defined time has elapsed, uncontrollable event *timeout* is generated. Free behaviour model G_3^g represents the message sending capability. Controllable events *sendBG* and *sendBB* enable the broadcasting message of available green and blue followers, respectively. Controllable events *sendOG* and *sendOB* are the messages emitted by a leader to offer membership in the group to a specific green or blue follower, respectively. Controllable events *sendAG* and *sendAB* are the messages that confirm the acceptance of the green or blue follower, respectively. Controllable event *sendE* causes followers that are already part of a group to relay the last received message that is related to its leader. Controllable event *msgStop* stops the sending of the current message. Free behaviour model G_4^g defines the message receiving capability. Controllable

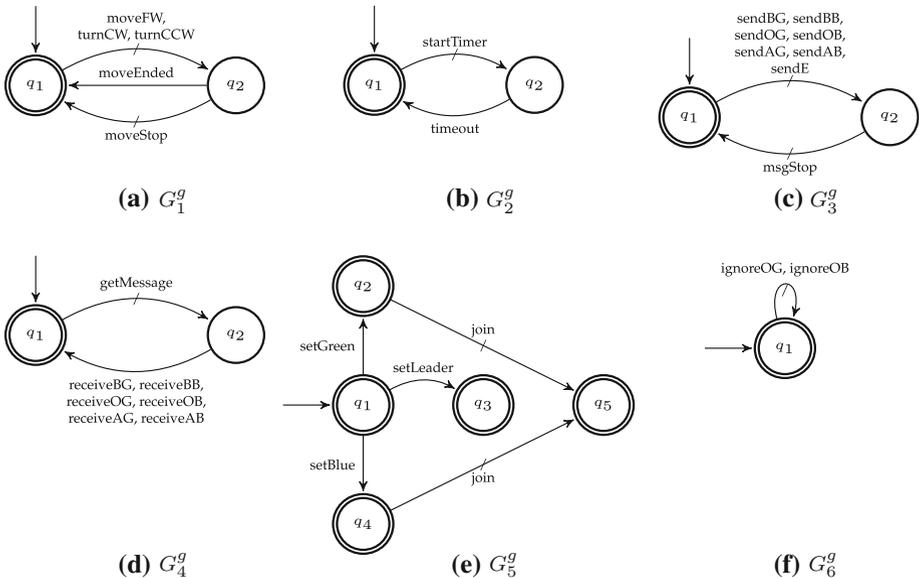


Fig. 10 Free behaviour models for the group formation case study. **a** Motion capabilities; **b** timer; **c** message transmission; **d** message reception; **e** the robot’s configurations; **f** the robot’s ability to choose not to make an offer

event *getMessage* reads the most recent received message in the buffer. Uncontrollable events *receiveBG* and *receiveBB* are triggered when a broadcast message of a blue or green follower is received; *receiveOG* and *receiveOB* are triggered when a message is received that offers membership to a green or blue follower, respectively; *receiveAG* and *receiveAB* are triggered when receiving an acceptance message by a green or blue follower, respectively, to join a group. Free behaviour model G_5^g defines the robot configuration subsystem. The robots can be configured as a leader by the uncontrollable event *setLeader*, or as a follower by *setGreen* or *setBlue*; it is not possible for a robot to change its configuration. The configuration is randomly selected during initialisation. Followers can join a group triggering the controllable event *join*. In the free behaviour model G_6^g , the leader can choose not to send an offer by the controllable events *ignoreOG* and *ignoreOB*. This is used to implement the equilibrium criterion.

Figure 11 illustrates the specifications for the group formation strategy. Specification E_1^g defines that followers that have not yet joined any group can move. All other robots are not allowed to move. Specification E_2^g defines that a leader robot can send an offer to a follower when it receives a corresponding broadcast message. Alternatively, it can choose to ignore the request. Specification E_3^g controls the follower’s cycle of messages. Robots broadcast their colour and identification code until they receive an offer for their colour. When this occurs, they send an acceptance message and join the group. When a follower joins a group, it relays the messages it receives, using an echo function triggered by controllable event *sendE*. Specification E_4^g controls the transmission mechanisms. A message is transmitted by the subsystem over a pre-defined period of time. Once a message is sent, it can only be stopped after the timeout of the subsystem. Specification E_5^g implements the equilibrium criterion for the leader. In state q_1 , the system is in equilibrium and can make offers to both classes of followers. In state q_2 , there is one more blue follower than green followers. Therefore, offers can only be made to green followers. State q_3 describes the equivalent situation where there are more green than blue followers. Finally, specification E_6^g defines the relay mode of a follower. Any received message is retransmitted after joining a group by event *sendE*. For all messages, an identification code filter is implemented to minimise the traffic load in a layer that links the abstract discrete events to the hardware (see Sect. 6); this is not implemented by using the formal method.

5 Supervisor synthesis

The supervisor represents the control logic of the robot. To obtain the supervisor, one has to restrict the free behaviour models according to the specifications. In other words, the robot should be allowed to perform only those actions that are compatible with the specifications. Formally, this is done by combining the free behaviour models and specifications using synchronous composition. The synchronous composition (represented by $\cdot||\cdot$) of two generators G_a and G_b with alphabet $\Sigma_i, i \in \{a, b\}$ is defined as

$$G_a||G_b = (Q_a \times Q_b, \Sigma_a \cup \Sigma_b, \delta_{a||b}, (q_{0a}, q_{0b}), Q_{m_a} \times Q_{m_b}), \tag{2}$$

where

$$\delta_{a||b}((q_a, q_b), e) = \begin{cases} (\delta_a(q_a, e), \delta_b(q_b, e)) & \text{if } \delta_a(q_a, e)! \wedge \delta_b(q_b, e)! \wedge e \in \Sigma_a \wedge e \in \Sigma_b \\ (\delta_a(q_a, e), q_b) & \text{if } \delta_a(q_a, e)! \wedge e \in \Sigma_a \wedge e \notin \Sigma_b \\ (q_a, \delta_b(q_b, e)) & \text{if } \delta_b(q_b, e)! \wedge e \notin \Sigma_a \wedge e \in \Sigma_b \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{3}$$

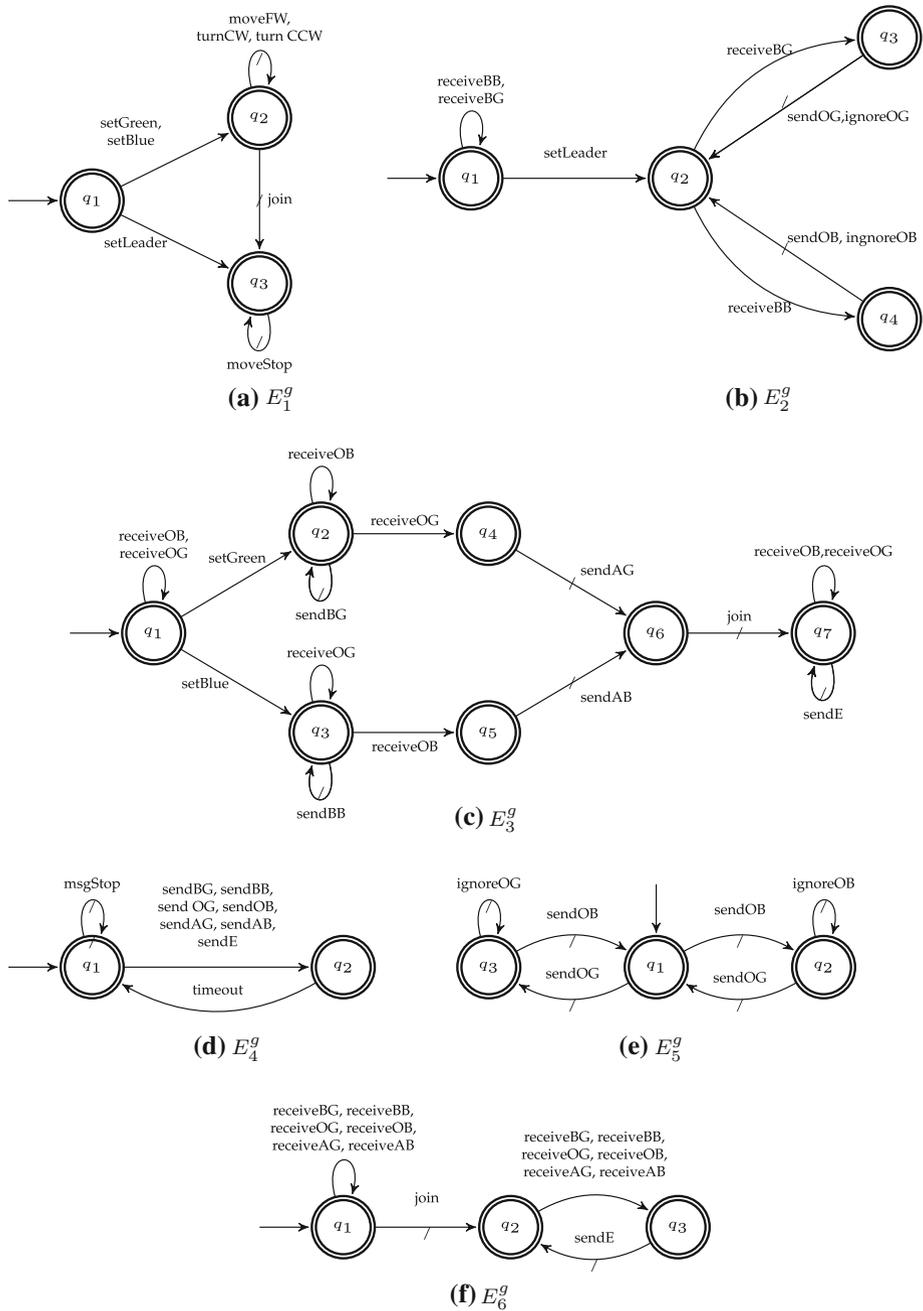


Fig. 11 Specification models for the group formation case study. **a** Allows followers that did not join a group to move; **b** allows leaders that received a broadcast to send or not to send an offer; **c** allows followers that received an offer message to join a group after sending an acceptance message; **d** ensures that a message is transmitted for a minimum period; **e** guarantees the equilibrium criterion; **f** the robot’s ability to choose not to make an offer

and $\delta(x, y)!$ means that δ is defined on input (x, y) . Equation 3 ensures that events that are not common to Σ_a and Σ_b can occur asynchronously, whereas events that are common to both alphabets must occur synchronously.

The synchronous composition of free behaviour models with specifications is called the target language. In the case of a single free behaviour model G and a single specification E , the target language K is defined as

$$K = G||E. \tag{4}$$

It is important to note that the target language is not necessarily controllable. A language K over an alphabet Σ is controllable with respect to the free behaviour model G and the set of uncontrollable events $\Sigma_u \subseteq \Sigma$, if (Cassandras and Lafortune 2008):

$$\forall s \in \overline{L(K)}, \forall e_u \in \Sigma_u, se_u \in L(G) \Rightarrow se_u \in \overline{L(K)}, \tag{5}$$

where \overline{L} denotes the *prefix-closure* of a language L , that is, $\overline{L} = \{s \in \Sigma^* : \exists t \in \Sigma^* \wedge st \in L\}$. In other words, if s is a prefix of a word of the language generated by K , $L(K)$, and e_u an uncontrollable event that is physically possible to occur after this sequence (i.e. $se_u \in L(G)$), then se_u must also be a prefix of a word in $L(K)$ (i.e. $se_u \in \overline{L(K)}$).

Let us consider controllability in more detail. Each state $q_{K(y)}$ of a target language $K = G||E$ can be mapped to a state $q_{G(x)}$ in G . $q_{K(y)}$ can be considered as a composed state $(q_{G(x)}, \cdot)$. If an event e is enabled in $q_{G(x)}$ but not in $q_{K(y)} = (q_{G(x)}, \cdot)$, it is physically possible to occur, but denied by the control specification. This corresponds to case “undefined” in Eq. 3. If e is an uncontrollable event, $q_{K(y)}$ is called a bad state, as the controller is not able to disable event e when the state is reached. The language is then uncontrollable. Thus, $q_{K(y)}$ is a bad state if:

$$\exists e \in \Sigma_u : e \in \Sigma_{G(x)} \text{ and } e \notin \Sigma_{K(y)}, \tag{6}$$

where $\Sigma_{G(x)}$ denotes the set of events defined in state $q_{G(x)}$ and $\Sigma_{K(y)}$ denotes the set of events defined in state $q_{K(y)}$. To extract the controllable sub-language from an uncontrollable language, all bad states (e.g. q_{bad}) and all states that have uncontrollable paths to any bad state (i.e. $q_a : \exists s \in \Sigma_u^+ : \delta(q_a, s) = q_{bad}$) are removed. The resulting language is minimally restrictive (Ramadge and Wonham 1987). In other words, it is the largest sub-language of K that is controllable.

Figure 12 shows an example of a bad state and its removal. The composition of G , Fig. 12a, and E , Fig. 12b, results in target language K , Fig. 12c. State $q_{(2,2)}$ in K is related to state q_2 in G where both uncontrollable events a and b are enabled, but a is disabled in q_2 of specification E , and hence it is disabled in $q_{(2,2)}$ in K . As a consequence, the uncontrollable event a could occur in $q_{(2,2)}$, even though it should not occur according to the specification. To prevent the event from occurring, state $q_{(2,2)}$ must not be reached. Therefore, it is removed. As the controller can only disable controllable events, it is necessary to remove also all states with an uncontrollable path to $q_{(2,2)}$, if any. Following the removal of bad state $q_{(2,2)}$, the target language, Fig. 12d, is controllable.

States of a generator that are reachable from initial state q_0 are called accessible. The initial state is accessible by definition. If all states of a generator are accessible, the generator is called accessible. States of a generator that can reach at least one marked state $q \in Q_m$ are called coaccessible. Marked states are coaccessible by definition. If all states of a generator are coaccessible, the generator is called coaccessible. All non-accessible and non-coaccessible states of G can be removed with the operator $\text{trim}(G)$. This operator guarantees that the corresponding language $L(G)$ is non-blocking (Wonham and Ramadge 1988). In other words, no deadlocks can occur.

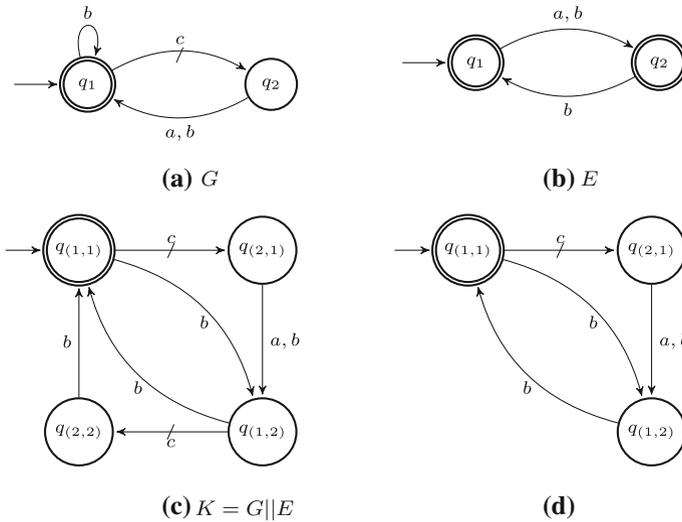


Fig. 12 Example of a bad state and its removal. The composition of G (a) and E (b) results in target language K (c). Removing bad state $q(2,2)$ results in a controllable language (d). For details, see text

Figure 13 shows an example of a non-coaccessible state and its removal by the trim operator. This example considers the controllable event a and the uncontrollable events b and c . The generator in Fig. 13a has all states accessible from initial state q_1 . States $q_1, q_2,$ and q_3 are also coaccessible, as all of them have a path to marked state q_1 . State q_4 is non-coaccessible and is eliminated (see Fig. 13b). However, in state q_3 the uncontrollable event c can occur which is not desirable (it previously led to the non-coaccessible state q_4). Thus, q_3 must be removed as well (Fig. 13c). Note that as only transitions triggered by controllable events led to q_3 , the resulting target language is controllable.

$SupC(G, K)$ removes bad states from language K (obtained by Eq. 4) and applies the trim operator to guarantee the accessibility and coaccessibility. It works iteratively as the trim can result in additional bad states. The remaining language,

$$L_m(S/G) = SupC(G, K), \tag{7}$$

is controllable, accessible, and coaccessible. As a consequence, it is non-blocking.

With the presented methods, supervisors can be synthesised that possess all the aforementioned properties. The supervisors can be represented as monolithic, modular, and local modular supervisors.

5.1 Monolithic supervisor

If all free behaviour and specification models are composed to a single supervisor S , S is called monolithic. The first step to synthesise a monolithic supervisor is to compose all m^θ free behaviour models in a single generator⁴:

$$G^\theta = G_1^\theta || \dots || G_{m^\theta}^\theta. \tag{8}$$

⁴ Note that θ denotes the case study as defined in Sect. 4.

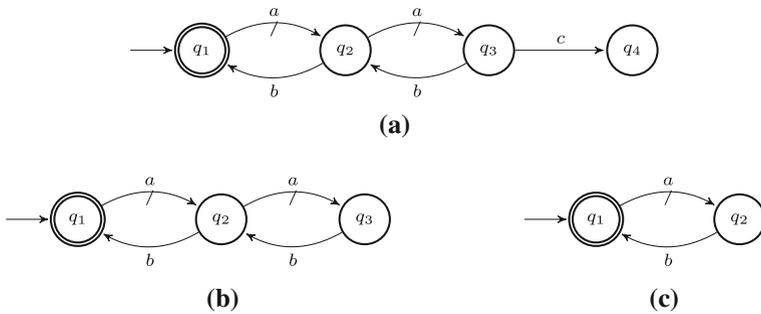


Fig. 13 Example of a non-coaccessible state and its removal by $\text{trim}(\cdot)$. In **a**, states q_1 , q_2 , and q_3 are coaccessible, as all of them have a path to marked state q_1 . State q_4 is not coaccessible and hence removed (**b**). As state q_4 could be reached from state q_3 through uncontrollable event c , q_3 is removed as well (**c**). State q_3 could only be reached through controllable events, which can be disabled by the supervisor

All n^θ specifications are also composed in a single generator:

$$E^\theta = E_1^\theta || \dots || E_{n^\theta}^\theta. \tag{9}$$

The monolithic target language, K^θ , is obtained by the synchronous composition of G^θ and E^θ :

$$K^\theta = G^\theta || E^\theta. \tag{10}$$

Finally, the monolithic supervisor S^θ is obtained as:

$$S^\theta : L_m(S^\theta / G^\theta) = \text{SupC}(G^\theta, K^\theta). \tag{11}$$

5.2 Modular supervisors

Due to the parallel composition, the number of states may grow exponentially with the number of free behaviour models and specifications. As a result, a prohibitively large amount of program memory can be required to store the control logic. To alleviate this problem, modular supervisors were proposed (Wonham and Ramadge 1988). The modular approach composes one supervisor for each specification. These supervisors can then be executed in parallel.

The free behaviour models are composed into a single generator $G^{\text{mod},\theta}$. This is done in the same way as for the monolithic approach (see Eq. 8). Thus, $G^{\text{mod},\theta} = G^\theta$.

Rather than calculating a single target language, one target language $K_j^{\text{mod},\theta}$ is calculated for each specification E_j^θ :

$$K_j^{\text{mod},\theta} = G^{\text{mod},\theta} || E_j^\theta \quad \forall j \in \{1, \dots, n^\theta\}. \tag{12}$$

The modular supervisor is obtained for each target language, analogous to Eq. 11:

$$S_j^{\text{mod},\theta} : L_m(S_j^{\text{mod},\theta} / G^{\text{mod},\theta}) = \text{SupC}(G^{\text{mod},\theta}, K_j^{\text{mod},\theta}) \quad \forall j \in \{1, \dots, n^\theta\}. \tag{13}$$

The modular approach requires the specifications to have no conflicts. To check for conflicts, all modular supervisors are composed together into $S_{||}^{\text{mod}}$. $S_{||}^{\text{mod}}$ is then compared with the monolithic supervisor. If they are not equivalent, that is, they produce different languages, then a conflict exists (see (Wonham and Ramadge 1988; Lopes et al. 2012), for details). Where a conflict occurs between specifications, the conflicting specifications have to be composed

together in a single supervisor. This reduces the number of supervisors. For example, if two specifications E_1 and E_2 are in conflict, then the supervisors S_1^{mod} and S_2^{mod} are replaced by $S_{1,2}^{mod}$, where $L_m(S_{1,2}^{mod}) = SupC(G, K_{1,2}^{mod})$ and $K_{1,2}^{mod} = G || E_1 || E_2$.

5.3 Local modular supervisors

The local modular approach (Queiroz and Cury 2000a, b, 2002) explores not only the modular property of specifications but also of free behaviour models. It reduces the number of free behaviour models used in the synthesis of each supervisor. This may result in supervisors with fewer states and transitions in total.

In the local modular approach, similar to the modular one, a supervisor is created for each control specification. However, only the free behaviour models that are affected by the particular control specification are taken into account. Thus, each specification E_j^{loc} has its own local free behaviour model G_j^{loc} , which is the parallel composition of all free behaviour G_i that have at least one event in common with E_j .

Table 1 shows the relation of events for each specification for the segregation case study. The local free behaviour models are obtained as:

$$\begin{aligned}
 G_1^{loc,s} &= G_1^s || G_2^s, \\
 G_2^{loc,s} &= G_2^s || G_3^s, \\
 G_3^{loc,s} &= G_3^s || \dots || G_6^s.
 \end{aligned}
 \tag{14}$$

The appendix gives the local free behaviour models for the remaining four case studies.

Table 1 Events used by the specifications and free behaviour models for the segregation case study

		E_1^s	E_2^s	E_3^s
G_1^s	<i>press</i>	✓		
G_2^s	<i>sendR</i>	✓	✓	
	<i>sendG</i>	✓	✓	
	<i>sendB</i>	✓	✓	
	<i>sendNothing</i>	✓	✓	
G_3^s	<i>moveFW</i>		✓	✓
	<i>turnCW</i>		✓	✓
	<i>turnCCW</i>		✓	✓
	<i>moveEnded</i>			✓
	<i>moveStop</i>			✓
G_4^s	<i>getR</i>			✓
	<i>getNotR</i>			✓
G_5^s	<i>getG</i>			✓
	<i>getNotG</i>			✓
G_6^s	<i>getB</i>			✓
	<i>getNotB</i>			✓
Local models		$G_1^{loc,s}$	$G_2^{loc,s}$	$G_3^{loc,s}$

In the local modular approach, only the relevant free behaviour models are used when composing a supervisor for a specification. These are the free behaviour models that have at least one event in common with the specification

As in the modular approach, one target language is calculated for each specification. However, each target language uses its own local free behaviour model:

$$K_j^{loc,\theta} = G_j^{loc,\theta} || E_j^\theta \quad \forall j \in \{1, \dots, n^\theta\}. \tag{15}$$

The local modular supervisor $S_j^{loc,\theta}$ is obtained for each target language $K_j^{loc,\theta}$, analogous to Eq. 13:

$$S_j^{loc,\theta} : L_m(S_j^{loc,\theta} / G_j^{loc,\theta}) = SupC(G_j^{loc,\theta}, K_j^{loc,\theta}) \quad \forall j \in \{1, \dots, n^\theta\}. \tag{16}$$

5.3.1 Enabled events in modular and local modular supervisors

In the case of multiple supervisors S_1, \dots, S_n running in parallel, a controllable event e_c is enabled if for every supervisor S_j , where $e_c \in \Sigma_j$, the transition function $\delta_j(q_j, e_c)$ is defined.

5.4 Comparison

Table 2 compares the three methods introduced in this section in terms of size of target language and supervisors for all case studies. In particular, it lists the number of states and transitions. These performance metrics are related to the program memory required to store the control strategy. The number of state transitions is based on the minimised version of each automaton (see Brzozowski (1962)). The local modular approach turned out to be the most memory efficient in three of the four case studies. In the remaining case study, it was almost on par with the best alternative. The modular approach is the least memory efficient for the considered cases.

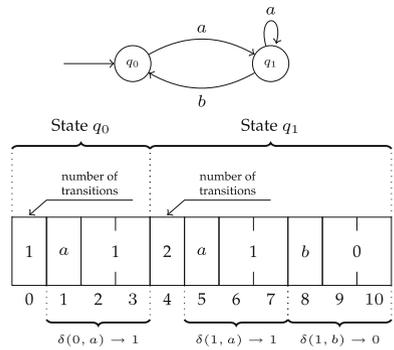
The local modular approach requires more effort when synthesising the supervisors. One needs to check whether conflicts occur. However, once the supervisors are obtained, as can be seen from Table 2, the local modular approach often outperforms the other approaches in terms of total number of states and transitions (and hence in memory usage).

Table 2 Total number of states and transitions for each case study when using monolithic, modular, and local modular synthesis approaches, respectively

		Monolithic		Modular		Local modular	
		K	S	K	S	K	S
Segregation	States	128	128	256	256	32	32
	Transitions	696	696	1720	1720	103	103
Aggregation	States	7 (9)	7 (9)	8	8	8	8
	Transitions	18 (22)	18 (22)	28	28	20	20
Object clustering	States	13 (16)	13 (16)	12	12	12	12
	Transitions	48 (57)	48 (57)	66	66	48	48
Group formation	States	280	130	560	488	93	79
		(304)	(138)	(776)	(680)	(108)	(92)
	Transitions	1446	531	6708	5768	553	452
		(1602)	(561)	(9272)	(8012)	(641)	(525)

Data correspond to the minimised version of the target language K and supervisor S . Data in parentheses show the numbers prior to minimisation, if different. The best results are highlighted in bold

Fig. 14 Supervisor data structure in memory (Lopes et al. 2012). The first element of each state represents the number of outgoing transitions. It is followed by blocks of three elements, which detail the event that triggers the transition and the resulting state



6 Implementation of supervisory control in swarm robotics

Our implementation is based on the SCT architecture proposed by (Queiroz and Cury 2000b, 2002). It adds three layers on top of the robot’s hardware: the supervisor (which is at the highest level), the generator player, and then the operational procedures. In this paper, we implement complete local modular supervisors.⁵

Our implementation uses the open source software tool Nadzoru (Lopes et al. 2012; Lopes 2012; Pinheiro et al. 2015) available at <http://www2.joinville.udesc.br/~gasr/nadzoru/>. Nadzoru supports the design of free behaviour models and control specifications, the synthesis of supervisors, and automatic code generation. Furthermore, we extended Nadzoru to support the e-puck and Kilobot platforms.

A video demonstrating the use of Nadzoru is available in the electronic supplementary material. The Nadzoru tool can be found together with all implementations in (Lopes et al. 2015).

6.1 Supervisor representation in memory

Figure 14 illustrates the data structure that stores the synthesised supervisor in memory (Lopes et al. 2012). Each state is represented as a part of this data structure. Each part describes all output transitions from that state. The first byte for each part is the amount of output transitions (o). It is followed by o sets of 3 bytes, where each set represents a transition. The first byte of each set represents the event. The other two bytes determine the target state. This data structure is limited to 256 events, 2^{16} states and 255 output transitions per state. The memory occupation (in bytes) of the supervisor data structure is given by:

$$mem = s + 3 \times t, \tag{17}$$

where s is the total number of states and t the total number of transitions among all supervisors. The method can be adapted to support higher numbers of events, states, and transitions (Lopes et al. 2012).

In addition, a vector of size e bytes stores whether events are controllable or uncontrollable, where e is the number of events used in the system. A matrix of size $e \times N$ bytes defines which events are part of each supervisor, where N is the number of supervisors. A vector of size $N \times 2$ bytes stores the current state of all supervisors.

⁵ Note that these differ from reduced local modular supervisors (Queiroz and Cury 2000b, 2002).

6.2 Generator player

The *generator player*—also called automata player—is a virtual machine. It executes the generators realising the supervisors. An arbitrary number of generators can run in parallel.

The generator player is given by Algorithm 1. Its logic stores the states of all generators. It checks whether any uncontrollable events occurred (by calling functions in the operational procedures, see Sect. 6.3). If an uncontrollable event occurred, the generators' states are updated accordingly. Otherwise, the generator player determines the list of enabled controllable events. If the list is not empty, the generator player selects one such event (e.g. at random) and updates the generators' states accordingly. It also calls functions in the operational procedures to perform the action associated with the event.

Algorithm 1 Generator player

```

1: Let  $N$  be the number of generators
2: procedure GENERATOR PLAYER
3:   for all  $j \in \{1, 2, \dots, N\}$  do
4:     set current state  $c_j$  to initial state  $q_{0j}$ ;
5:   end for
6:   while true do
7:     if uncontrollable event  $e_u \in \Sigma_u$  occurred then
8:       for all  $j \in \{1, 2, \dots, N\}$  do
9:          $c_j = \delta_j(c_j, e_u)$ ;
10:      end for
11:     else
12:       calculate the set of enabled controllable events  $\psi(c_1, c_2, \dots, c_n)$ ;
13:       if  $\psi(c_1, c_2, \dots, c_n) \neq \emptyset$  then
14:         select one controllable event  $e_c \in \psi(c_1, c_2, \dots, c_n)$ ;
15:         for all  $j \in \{1, 2, \dots, N\}$  do
16:            $c_j = \delta_j(c_j, e_c)$ ;
17:         end for
18:         execute callback function of  $e_c$ ;
19:       end if
20:     end if
21:   end while
22: end procedure

```

To perform the experiments, the generator player was implemented for both swarm platforms—Kilobot and e-puck. These implementations (one per target platform) were intensively used across the case studies and hence can be considered reliable. This is an inherent advantage of virtual machines. It reduces the code that has to be manually validated to the operational procedures, which link the abstract events to the hardware.

6.3 Operational procedures

The operational procedures are a low-level interface of the controller to the hardware (Queiroz and Cury 2002). Operational procedures were originally designed for manufacturing environments. As a result, they were mainly used to translate events to signals on pins of programmable control logic devices and vice versa. In the following, we show how to use operational procedures in a more unrestricted way. Nadzoru allows the developer to define callback functions in a separate file or to input the code in the tool, which then outputs the complete final code.

The operational procedures, implemented by the developer, define one callback function for each event. For controllable events, the generator player calls this function to perform an action (see line 18 of Algorithm 1). In the segregation case study, for example, controllable event *turnCW* is used to turn the robot clockwise for a random duration of time. The corresponding callback function could be realised, using functions from the e-puck library, as follows:

```
void callback_turnCW( void* data ){
    int t = 5 + (rand()
    dirLeft = -1; dirRight = 1;
    //set wheels speed
    e_set_speed_left(500);
    e_set_speed_right(-250);
    //set required number of steps for stepper motor counter
    e_set_steps_left(-500*t);
    e_set_steps_right(250*t);
}
```

For uncontrollable events, the generator player calls a function to determine whether these events occurred (see line 7 of Algorithm 1). For example, for uncontrollable event *moveEnded*, the callback function could check whether the aforementioned duration has elapsed:

```
unsigned char moveEnded( void* data ){
    return e_get_steps_left()*dirLeft < 0 || e_get_steps_right()*dirRight < 0;
}
```

6.4 Memory usage

Table 3 shows the memory usage of the control software using the proposed implementation for each case study and robot platform. The usage is broken down into four components: libraries/base code, operational procedures, generator player, and supervisors. Libraries are a group of generic routines or resources that are available to be used in any program. Base code includes initialisation routines not related to the operational procedures.

For all case studies and robot platforms, the memory usage is within 10–30 kB. Only a small fraction of this can be attributed to the operational procedures, generator player, and supervisors. The use of memory to store the supervisor is, however, larger than the theoretical amount derived in Sect. 6.1; the values depend on the specific overhead and implementation details of each compiler.

Table 3 Memory usage of the control software for each case study and robot platform

Case	Compiler/ platform	Libraries/ base code	Operational procedures	Generator player	Super- visors
Segregation	XC16/e-puck	25,308	1773	2106	654
Segregation	WinAVR/Kilobot	10,968	1512	1478	418
Aggregation	XC16/e-puck	17,502	408	2016	207
Object clustering	XC16/e-puck	16,695	696	2004	378
Group formation	WinAVR/Kilobot	10,968	1742	1584	1704

Values are in bytes

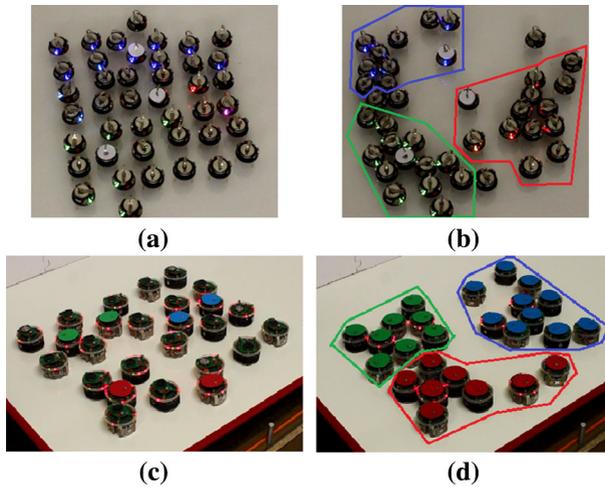


Fig. 15 Snapshots from a segregation trial with Kilobots: **a** initial grid formation with three leaders, marked with white tags; **b** result after segregation occurred. Trial with e-pucks: **c** initial grid formation with three pairs of leaders marked with tags; **d** result after segregation occurred, tags were added after the experiments for visualisation

7 Experiments

This section describes the experiments that we performed using local modular supervisors. The experiments are used to validate our implementation of SCT in practice. In particular, they test whether the modelled specifications match with the synthesised control logic, as observed during the trials. The electronic supplementary material offers a selection of video recordings. Video recordings from all 50 experimental trials and additional resources (models, the Nadzoru tool, the used source code) can be found in (Lopes et al. 2015).

7.1 Segregation experiment

The segregation experiment took place in a two-dimensional $1.20 \text{ m} \times 0.90 \text{ m}$ arena. It used a group of 39 follower robots in the Kilobot experiment and 20 follower robots in the e-puck experiment. These robots were initially distributed on a grid. Three leader robots were added inside the grid in the Kilobot experiment. For the e-puck experiment, three pairs of leaders were added; this was done as the infrared (IR) signal range of the e-puck is small in relation to its size. For each robot platform, ten trials were performed for 300 s or until the robots were segregated, whichever occurred first. The robots were considered to be segregated if they all receive a signal of only one leader or no signal at all—as indicated by their light-emitting diodes (LEDs). Visual inspection confirmed that the SCT controller performed the segregation task as intended. Figure 15 shows snapshots taken from one of the experimental trials with the Kilobots and e-pucks, respectively. Figure 15a, c shows the initial grid formation with leaders marked using tags. Figure 15b, d shows the result after segregation occurred. For Fig. 15d, tags were added to all robots after the experiments for visualisation purposes (based on the robots' states as indicated by their LEDs).

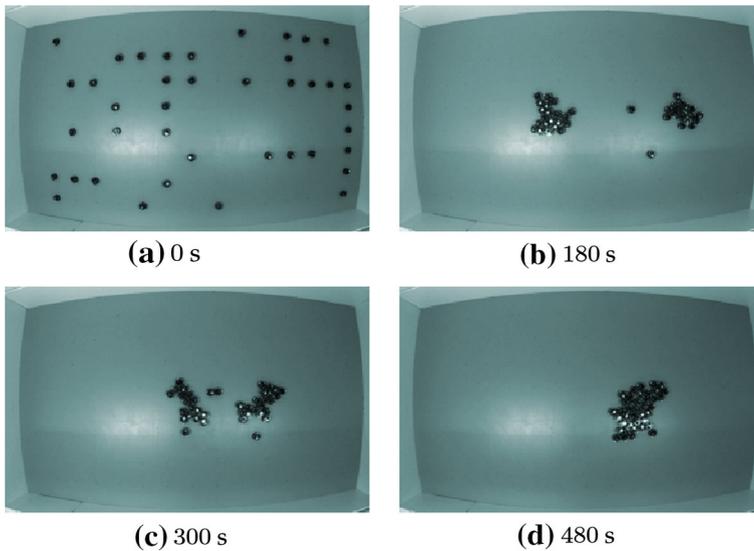


Fig. 16 Sequence of snapshots taken from one of the trials where 40 e-pucks perform the aggregation task. **a** The initial positions of the robots. **b–d** The experiment after 120, 300, and 480 s

7.2 Aggregation experiment

The aggregation experiment used the same configuration as (Gauci et al. 2014a). The e-puck robot used its on-board camera to determine the type of object within its line of sight. The camera is a CMOS RGB colour camera with a resolution of 640×480 and a field of view of $56^\circ \times 42^\circ$. To simplify identification, the robots were fitted with black skirts. Trials were performed in a $400 \text{ cm} \times 225 \text{ cm}$ light grey floor arena surrounded by white walls that were 50 cm in height. The arena had 120 pencil marks distributed as a 15×8 grid with columns and rows spaced 25 cm apart. Forty robots were uniformly randomly distributed over the marks in the arena.

In principle, one single pixel from the centre of the camera would be enough to realise the line-of-sight sensor. To account for misalignment between the orientations of the robots' cameras, the sensor was implemented as a column of pixels (for details, see (Gauci et al. 2014a)). This provided reliable readings in a range of up to 150 cm (Gauci et al. 2014a).

Ten trials were performed, each lasting 900 s. An IR signal was emitted to instruct the robots to turn in place for a random period of time. As a result, the robots were randomly orientated. A second IR signal instructed the robots to start the controller. In case of failure of any robot (e.g. the robot reset because of a collision or low battery), a restart of the controller was attempted via IR signal.

Figure 16 shows snapshots taken from one of the experimental trials, where 40 e-pucks were performing the aggregation task.

In one trial, we tried—without success—to manually reset a robot with hardware failure. In five of the ten cases, one to three robots of the forty became unresponsive. The remaining robots achieved aggregation in all ten trials.

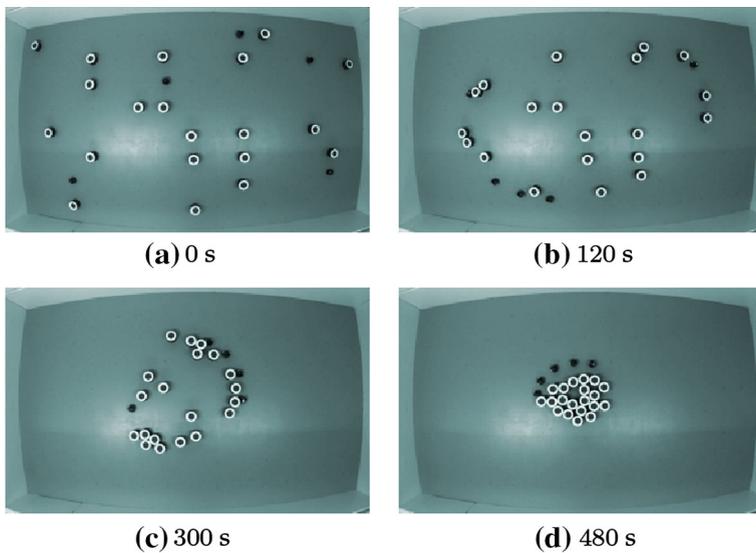


Fig. 17 Sequence of snapshots taken from one of the trials in which 5 e-pucks—in *black*—cluster 20 objects—in *white*. **a** The initial positions of the robots and the objects. **b–d** The experiment after 120, 300, and 480 s

7.3 Object clustering experiment

The object clustering experiment used the same configuration as in (Gauci et al. 2014b). In particular, the robots were fitted with green skirts and the objects used were red cylinders with 10 cm diameter and 10 cm height. As for the aggregation case study, the camera was used as a theoretical one-pixel sensor to determine the type of object within the line of sight. The implementation of the camera sensor is detailed in (Gauci et al. 2014b). Trials were performed in the same arena and using the same conditions as in the aggregation case study.

Ten trials were performed, each lasting 600 s. As in (Gauci et al. 2014b), 5 e-pucks and 20 objects were used. Figure 17 shows snapshots taken from one of the experimental trials.

To evaluate whether the performance of object clustering using SCT is similar to the original implementation (Gauci et al. 2014b), we used two metrics to characterise the performance. We measured the proportion of objects in the largest cluster and the compactness of objects. Two objects are considered to belong to the same cluster if there is a sequence of objects connecting them, such that any adjacent objects are no more than 10 cm apart. The compactness of objects ($u^{(t)}$) is defined as (Gauci et al. 2014b):

$$u^{(t)} = \frac{1}{4r_o^2} \sum_{i=1}^{N_o} \|\mathbf{p}_i^{(t)} - \bar{\mathbf{p}}^{(t)}\|^2, \quad (18)$$

where r_o is the radius of the object, N_o is the number of objects, $\mathbf{p}_i^{(t)}$ denotes the position of the object i , and $\bar{\mathbf{p}}^{(t)}$ represents the centroid of the centre of the objects.

The clustering dynamics are plotted in Fig. 18. The coloured curves correspond to the 10 trials, and the black dashed line represents the mean. Figure 18a, b shows the proportion of objects in the largest cluster. The results from (Gauci et al. 2014b) are plotted in (a), and those obtained using SCT in (b). Figure 18c, d shows the compactness of objects. The results from (Gauci et al. 2014b) are plotted in (c), and those obtained using SCT in (d). Overall the

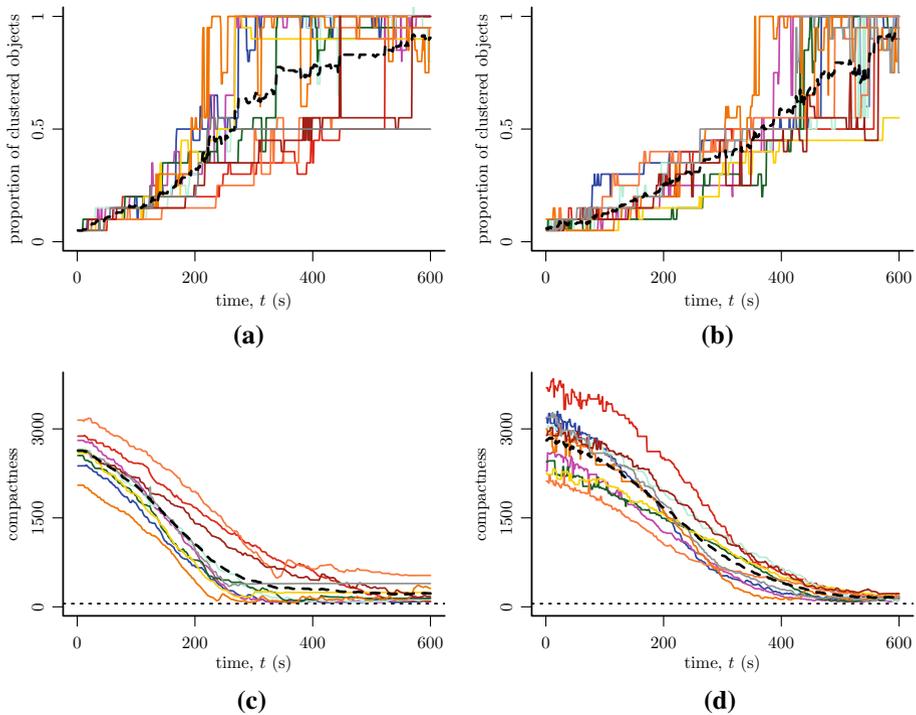


Fig. 18 Dynamics of object clustering with 5 e-pucks and 20 objects. **a** and **c** are plotted using data from (Gauci et al. 2014b). **b** and **d** present the results for the controller that was synthesised using SCT. In **a** and **b**, the relationship between the proportion of clustered objects and time is displayed. **c** and **d** display the compactness of objects in relation to time. Each coloured line represents one experimental trial. The thick black dashed line indicates the mean. The horizontal dotted black line indicates a theoretical lower bound of the compactness for 20 objects (Gauci et al. 2014b)

results are similar, and in both implementations the robots succeeded in clustering the objects. We notice that in the first half of the experiment—0 to 300 s—the original method had a slightly faster convergence compared to the use of SCT; in the second half, SCT overcomes that difference.

7.4 Group formation experiment

To test the scalability of the approach, 600 Kilobots were used in the group formation experiment. The experiment took place in a 2.20 m \times 2.20 m arena with a glass surface. The robots were uniformly distributed over the arena. The trial duration was limited to 600 s. The robots were started using an overhead programmer (OHP) (Rubenstein et al. 2012). The OHP can only communicate in a radius of around 50 cm. Due to the size of the arena, the robots could not all be started at the same time. It took approximately 60 s to initialise all the robots.

In total, 10 experimental trials were conducted. While it is difficult to monitor the continuous operation of 600 autonomous robots, visual inspection confirmed that they formed the groups as intended. Figure 19 shows snapshots taken from one experimental trial. A video recording is included in the electronic supplementary material. Video recordings from all experimental trials can be found in (Lopes et al. 2015). Note that we use the robot's LED to

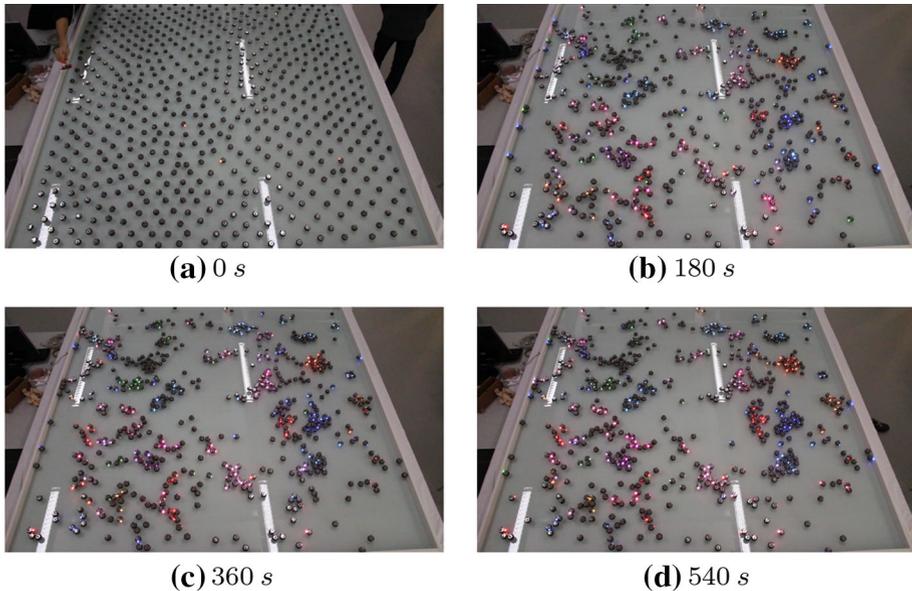


Fig. 19 Sequence of snapshots taken from one of the trials where 600 Kilobots perform the group formation task. **a** The robots in their initial position and **b–d** show the positions and states of robots after 180, 360, and 540 s

indicate its type. Leaders are represented by non-blinking randomly chosen colours. When a follower joins a group, it changes its colour to match the leader. Recall that there are two types of followers and that an equal number of them (± 1) will join each particular leader. To visually discriminate between followers of different types even after they join a leader, they flash their LEDs with two distinct frequencies.

8 Conclusions

This paper proposed and demonstrated the use of supervisory control theory (SCT) for formally developing controllers in swarm robotics. Using a series of case studies, it illustrated how to formally model the capabilities of robots and their desired behaviour (specifications). Supervisors—controllers in the form of formal languages—can then be derived from these models. The supervisors, here represented as regular languages, are driven through uncontrollable and controllable events. Uncontrollable events are triggered externally, for example, by the robot’s sensors. Controllable events are triggered to initiate an action, for example, to move the robot forward. The SCT design process reduces the action space to only those controllable events that do not violate the specifications. It guarantees that the supervisors are ‘controllable’ and ‘deadlock-free’. In addition, the supervisors can be subjected to a range of formal analysis tools (Akesson et al. 2006; Reiser et al. 2006; Rudie 2006; Feng and Wonham 2006).

Compared with other work on formal methods in swarm robotics, our work has the advantage that the control software is automatically derived from the problem specification. The open source software tool Nadzoru supports users through all stages of the development process, from specification to control software (for a demonstration see video in electronic

supplementary material). We extended Nadzoru to allow automatic code generation for two robot platforms—the Kilobot and the e-puck. The supervisors run on a virtual machine on-board each robot. The same supervisors can run on multiple platforms, further enhancing reusability. The only platform-specific source code the user would have to provide are the callback functions for events. For uncontrollable events, these would test whether the events occurred; for controllable events, these would execute the associated actions. Note that events can be reused for implementing solutions to different tasks, further reducing the amount of ad hoc development.

The case studies, which we reported, demonstrate that SCT is a promising method to generate state-of-the-art solutions for canonical tasks in swarm robotics. The tasks required the robots to gather, manipulate objects, and organise into logical groups. Systematic experiments with up to 40 e-pucks and up to 600 Kilobots confirmed the correctness of the implementation.

A limitation of SCT is that it assumes that the system under investigation can be represented as a discrete event system. In addition, the system and specifications have to be modelled as a formal language. To assist this process, graphical software tools (such as Nadzoru) can be used. The control logic can assume any behaviour that can be expressed using a regular language (Chomsky Type-3 grammar). Regular languages are realised by deterministic finite state machines (FSM), which are commonly used by designers of swarm robotics systems. Traditionally, the designer creates a single, relatively complex, FSM to express the desired behaviour. The supervisors used in this paper are an equivalent representation of such FSM. But instead of designing one complex FSM or supervisor, SCT can be used to decompose the behaviour into smaller and simpler parts and to separate the components related to the robot's abilities from those related to the specifications. This allows the designer to focus on each aspect individually. In principle, SCT can be used with formal languages of higher computational power. For example, deterministic Petri-nets or pushdown automata (Chomsky Type-2 grammar) can offer elegant solutions for problems involving unbounded variables (e.g. for robots counting their neighbours). Note, however, that some formal methods are not suitable for the control of DES (Sreenivas 1993) or may require alteration (Lacerda and Lima 2014).

In the future, we will investigate how to prove further properties of controllers modelled by SCT. We will also attempt to distribute a single supervisor across multiple robots.

Acknowledgments This research was supported by the Engineering and Physical Sciences Research Council (Grant No. EP/J013714/1 and EP/K031600/1). Y.K. Lopes acknowledges support by Coordination for the Improvement of Higher Education Personnel (CAPES)—Brazil (Grant Number: 0462/12-8). S.M. Trenkwalder is a recipient of a DOC Fellowship of the Austrian Academy of Sciences. We are thankful for the assistance provided by F. Perez-Diaz and G. Kapellmann while performing the experiments with the Kilobots.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

9 Appendix

This section presents the local free behaviour model calculations for the three case studies that were omitted in Sect. 5.3: aggregation, object clustering, and group formation.

Table 4 Events used by specifications and free behaviour models for the aggregation case study

		E_1^a	E_2^a	E_3^a	E_4^a
G_1^a	S_0	✓	✓		
	S_1	✓	✓		
G_2^a	V_0	✓		✓	✓
	V_1		✓	✓	✓
Local models		$G_1^{loc,a}$	$G_2^{loc,a}$	$G_3^{loc,a}$	$G_4^{loc,a}$

Table 5 Events used by specifications and free behaviour models for the object clustering case study

		E_1^c	E_2^c	E_3^c	E_4^c	E_5^c	E_6^c
G_1^c	S_0	✓	✓	✓			
	S_1	✓	✓	✓			
	S_2	✓	✓	✓			
G_2^c	V_0	✓			✓	✓	✓
	V_1		✓		✓	✓	✓
	V_2			✓	✓	✓	✓
Local models		$G_1^{loc,c}$	$G_2^{loc,c}$	$G_3^{loc,c}$	$G_4^{loc,c}$	$G_5^{loc,c}$	$G_6^{loc,c}$

9.1 Aggregation

Table 4 shows the relation of events for each specification for the aggregation case study. The local free behaviour models are obtained as:

$$\begin{aligned}
 G_1^{loc,a} &= G_2^{loc,a} = G_1^a || G_2^a, \\
 G_3^{loc,a} &= G_4^{loc,a} = G_2^a.
 \end{aligned}
 \tag{19}$$

9.2 Object clustering

Table 5 shows the relation of events for each specification for the object clustering case study. The local free behaviour models are obtained as:

$$\begin{aligned}
 G_1^{loc,c} &= G_2^{loc,c} = G_3^{loc,c} = G_1^c || G_2^c, \\
 G_4^{loc,c} &= G_5^{loc,c} = G_6^{loc,c} = G_2^c.
 \end{aligned}
 \tag{20}$$

9.3 Group formation

Table 6 shows the relation of events for each specification for the group formation case study. The local free behaviour models are obtained as:

$$\begin{aligned}
 G_1^{loc,g} &= G_1^g || G_5^g, \\
 G_2^{loc,g} &= G_3^g || \dots || G_6^g, \\
 G_3^{loc,g} &= G_6^{loc,g} = G_3^g || G_4^g || G_5^g, \\
 G_4^{loc,g} &= G_2^g || G_3^g, \\
 G_5^{loc,g} &= G_3^g || G_6^g.
 \end{aligned}
 \tag{21}$$

Table 6 Events used by specifications and free behaviour models for the group formation case study

		E_1^g	E_2^g	E_3^g	E_4^g	E_5^g	E_6^g
G_1^g	<i>moveFW</i>	✓					
	<i>turnCW</i>	✓					
	<i>turnCCW</i>	✓					
	<i>moveEnded</i>						
	<i>moveStop</i>	✓					
G_2^g	<i>startTimer</i>						
	<i>timeout</i>				✓		
G_3^g	<i>msgStop</i>				✓		
	<i>sendE</i>			✓	✓		✓
	<i>sendBG</i>			✓	✓		
	<i>sendBB</i>			✓	✓		
	<i>sendOG</i>		✓		✓	✓	
	<i>sendOB</i>		✓		✓	✓	
	<i>sendAG</i>			✓	✓		
	<i>sendAB</i>			✓	✓		
G_4^g	<i>getMessage</i>						
	<i>receiveBG</i>		✓				✓
	<i>receiveBB</i>		✓				✓
	<i>receiveOG</i>			✓			✓
	<i>receiveOB</i>			✓			✓
	<i>receiveAG</i>						✓
G_5^g	<i>receiveAB</i>						✓
	<i>setLeader</i>	✓	✓				
	<i>setGreen</i>	✓		✓			
	<i>setBlue</i>	✓		✓			
	<i>join</i>	✓		✓			✓
G_6^g	<i>ignoreOG</i>		✓			✓	
	<i>ignoreOB</i>		✓			✓	
Local models		$G_1^{loc,g}$	$G_2^{loc,g}$	$G_3^{loc,g}$	$G_4^{loc,g}$	$G_5^{loc,g}$	$G_6^{loc,g}$

References

- Akesson, K., Fabian, M., Flordal, H., & Malik, R. (2006). Supremica—An integrated environment for verification, synthesis and simulation of discrete event systems. In *2006 IEEE 8th international workshop on discrete event systems* (pp. 384–385). Piscataway, NJ: IEEE.
- Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., & Pappas, G. (2007). Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, *14*(1), 61–70.
- Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41.
- Brambilla, M., Brutschy, A., Dorigo, M., & Birattari, M. (2015). Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking. *ACM Transaction on Autonomous and Adaptive Systems*, *9*(4), 17:1–17:28.
- Brzozowski, J. (1962). Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*, *12*, 529–561.
- Cassandras, C. G., & Lafortune, S. (2008). *Introduction to Discrete Event Systems* (2nd ed.). New York: Springer.
- Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124.
- Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, *2*(2), 137–167.
- Costelha, H., & Lima, P. (2008). Modelling, analysis and execution of multi-robot tasks using Petri nets. In *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems (AAMAS'08)* (Vol. 3, pp. 1187–1190). Richland, SC: IFAAMS.
- Cowley, A., & Taylor, C. (2007). Orchestrating concurrency in robot swarms. In *Proceedings of the IEEE/RJS international conference on intelligent robots and systems* (pp. 945–950). Piscataway, NJ: IEEE.
- Dixon, C., Winfield, A., & Fisher, M. (2011). Towards temporal verification of emergent behaviours in swarm robotic systems. In R. Groß, L. Alboul, C. Melhuish, M. Witkowski, T. Prescott, & J. Penders (Eds.), *Towards autonomous robotic systems, volume 6856 of Lecture notes in computer science* (pp. 336–347). Berlin: Springer.
- Dixon, C., Winfield, A. F., Fisher, M., & Zeng, C. (2012). Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, *60*(11), 1429–1441.
- Emerson, E. (1990). Temporal and modal logic. In J. van Leeuwen (Ed.), *Handbook of theoretical computer science* (pp. 996–1072). Amsterdam: Elsevier.
- Fabian, M., & Hellgren, A. (1998). PLC-based implementation of supervisory control for discrete event systems. In *1998 IEEE 37th conference on decision and control* (Vol. 3, pp. 3305–3310). Piscataway, NJ: IEEE.
- Feng, L., & Wonham, W. M. (2006). TCT: A computation tool for supervisory control synthesis. In *2006 IEEE 8th international workshop on discrete event systems* (pp. 388–389). Piscataway, NJ: IEEE.
- Fierro, R., Das, A., Kumar, V., & Ostrowski, J. (2001). Hybrid control of formations of robots. In *Proceedings of ICRA 2001, IEEE international conference on robotics and automation* (pp. 157–162). Piscataway, NJ: IEEE.
- Forschelen, S., van de Mortel-Fronczak, J., Su, R., & Rooda, J. (2012). Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Systems*, *22*(4), 511–540.
- Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., et al. (2014a). An experiment in automatic design of robot swarms: Automode-vanilla, evostick, and human experts. In M. Dorigo, et al. (Eds.), *Swarm intelligence, ANTS 2014, volume 8667 of LNCS* (pp. 25–37). Berlin: Springer.
- Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., et al. (2015). Automode-Chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, *9*(2–3), 125–152.
- Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., & Birattari, M. (2014b). Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, *8*(2), 89–112.
- Gauci, M., Chen, J., Li, W., Dodd, T. J., & Groß, R. (2014a). Self-organised aggregation without computation. *The International Journal of Robotics Research*, *33*(9), 1145–1161.
- Gauci, M., Chen, J., Li, W., Dodd, T. J., & Groß, R. (2014b). Clustering objects with robots that do not compute. In *Proceedings of the 2014 international conference on autonomous agents and multi-agent systems (AAMAS '14)* (pp. 421–428). Richland, SC: IFAAMS.
- Gordon-Spears, D., & Kiriakidis, K. (2004). Reconfigurable robot teams: Modeling and supervisory control. *IEEE Transactions on Control Systems Technology*, *12*(5), 763–769.

- King, J., Pretty, R., & Gosine, R. (2003). Coordinated execution of tasks in a multiagent environment. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 33(5), 615–619.
- Knight, J. C., DeJong, C. L., Gobble, M. S., & Nakano, L. G. (1997). Why are formal methods not used more widely? In *Fourth NASA formal methods workshop* (pp. 1–12). NASA.
- Lacerda, B., & Lima, P. U. (2014). On the notion of uncontrollable marking in supervisory control of Petri nets. *IEEE Transactions on Automatic Control*, 59(11), 3069–3074.
- Liu, J., & Darabi, H. (2002). Ladder logic implementation of Ramadge-Wonham supervisory controller. In *2002 IEEE 6th international workshop on discrete event systems* (pp. 383–389). Piscataway, NJ: IEEE.
- Lopes, Y. K. (2012). Integração dos níveis MES, SCADA e controle da planta de manufatura com base na teoria de linguagens e autômatos. Master's thesis, Santa Catarina State University, Departamento de Engenharia Elétrica, Joinville, Brazil (in Portuguese).
- Lopes, Y. K., Leal, A. B., Rosso, R. S. U., & Harbs, E. (2012). Local modular supervisory implementation in microcontroller. In *Proceedings of the 9th international conference of modeling, optimization and simulation (MOSIM 2012)*.
- Lopes, Y. K., Leal, A. B., Dodd, T. J., & Groß, R. (2014). Application of supervisory control theory to swarms of e-puck and Kilobot robots. In M. Dorigo, et al. (Eds.), *Swarm Intelligence, ANTS 2014, volume 8667 of LNCS* (pp. 62–73). Berlin: Springer.
- Lopes, Y. K., Trenkwalder, S. M., Leal, A. B., Dodd, T. J., & Groß, R. (2015). Online supplementary material. <http://naturalrobotics.group.shef.ac.uk/supp/2015-001/>.
- Martinoli, A., Easton, K., & Agassounon, W. (2004). Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *The International Journal of Robotics Research*, 23(4–5), 415–436.
- Mass, D. G. N., Pinotti, A. J., & Leal, A. B. (2012). Síntese e implementação de controle supervisorio monolítico para um ice maker. *Anais do XIX Congresso Brasileiro de Automática, CBA, 19*, 5294–5301 (in Portuguese).
- Massink, M., Brambilla, M., Latella, D., Dorigo, M., & Birattari, M. (2013). On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics. *Swarm Intelligence*, 7(2–3), 201–228. doi:10.1007/s11721-013-0079-6.
- McNew, J. M., & Klavins, E. (2006). Locally interacting hybrid systems with embedded graph grammars. In *2006 45th IEEE conference on decision and control* (pp. 6080–6087). Piscataway, NJ: IEEE.
- McNew, J. M., Klavins, E., & Egerstedt, M. (2007). Solving coverage problems with embedded graph grammars. In A. Bemporad, et al. (Eds.), *Hybrid systems: Computation and control, volume 4416 of LNCS* (pp. 413–427). Berlin: Springer.
- Mesquita, A. (2010). *Exploiting stochasticity in multi-agent systems*. PhD thesis, University of California, Santa Barbara, CA.
- Mesquita, A. R., & Hespanha, J. P. (2012). Jump control of probability densities with applications to autonomous vehicle motion. *IEEE Transactions on Automatic Control*, 57(10), 2588–2598.
- Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klapocz, A., et al. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions* (Vol. 1, pp. 59–65).
- Pinheiro, L. P., Lopes, Y. K., Leal, A. B., & Rosso, R. S. U. (2015). Nadzor: A software tool for supervisory control of discrete event systems. In *Proc. of the 5th international workshop on dependable control of discrete systems (DCDS)* (Vol. 5).
- Queiroz, M. H., & Cury, J. E. R. (2000a). Modular supervisory control of large scale discrete event systems. In *Proceedings of international workshop on discrete event systems (WODES)* (pp. 103–110). Berlin: Springer.
- Queiroz, M. H., & Cury, J. E. R. (2000b). Modular control of composed systems. In *Proceedings of the 2000 american control conference* (pp. 4051–4055). Piscataway, NJ: IEEE.
- Queiroz, M. H., & Cury, J. E. R. (2002). Synthesis and implementation of local modular supervisory control for a manufacturing cell. In *Proceedings of 6th international workshop on discrete event systems (WODES)* (pp. 103–110). Piscataway, NJ: IEEE.
- Ramadge, P. J., & Wonham, W. M. (1987). Supervisory control of a class of discrete event process. *SIAM Journal on Control and Optimization*, 25(1), 206–230.
- Ramadge, P. J., & Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98.
- Reiser, C., Cunha, A. E. C., & Cury, J. E. R. (2006). The environment Grail for supervisory control of discrete event systems. In *2006 IEEE 8th international workshop on discrete event systems* (pp. 390–391). Piscataway, NJ: IEEE.
- Rubenstein, M., Ahler, C., & Nagpal, R. (2012). Kilobot: A low cost scalable robot system for collective behaviors. In *Proceedings of ICRA 2012* (pp. 3293–3298). Piscataway, NJ: IEEE.

- Rudie, K. (2006). The integrated discrete-event systems tool. In *2006 IEEE 8th international workshop on discrete event systems* (pp. 394–395), Piscataway, NJ: IEEE.
- Silva, D., Santos, E., Vieira, A., & de Paula, M. (2008). Application of the supervisory control theory in the project of a robot-centered, variable routed system controller. In *2008 IEEE international conference on emerging technologies and factory automation* (pp. 751–758). Piscataway, NJ: IEEE.
- Sreenivas, R. (1993). On a weaker notion of controllability of a language k with respect to a language l . *IEEE Transactions on Automatic Control*, *38*(9), 1446–1447.
- Tanner, H. G., Jadbabaie, A., & Pappas, G. J. (2007). Flocking in fixed and switching networks. *IEEE Transactions on Automatic Control*, *52*(5), 863–868.
- Tomlin, C., Pappas, G., Kosecka, J., Lygeros, J., & Sastry, S. (1998). Advanced air traffic automation: A case study in distributed decentralized control. In *Proceedings of the workshop control problems in robotics and automation* (pp. 261–295). Berlin: Springer.
- Tsalatsanis, A., Yalcin, A., & Valavanis, K. (2009). Optimized task allocation in cooperative robot teams. In *Proceedings of the 17th Mediterranean conference on control and automation (MED'09)* (pp. 270–275). Piscataway, NJ: IEEE.
- Tsalatsanis, A., Yalcin, A., & Valavanis, K. P. (2012). Dynamic task allocation in cooperative robot teams. *Robotica*, *30*(5), 721–730.
- Winfield, A. F. T., Sa, J., Fernández-Gago, M.-C., Dixon, C., & Fisher, M. (2005). On formal specification of emergent behaviours in swarm robotic systems. *International Journal of Advanced Robotic Systems*, *2*(4), 363–370.
- Wonham, W. M., & Ramadge, P. J. (1988). Modular supervisory control of discrete event system. *Mathematics of Control, Signals and Systems*, *1*(1), 13–30.
- Zavlanos, M., Tanner, H., Jadbabaie, A., & Pappas, G. (2009). Hybrid control for connectivity preserving flocking. *IEEE Transactions on Automatic Control*, *54*(12), 2869–2875.