Application of Supervisory Control Theory to Swarms of e-puck and Kilobot Robots

Yuri K. Lopes[1], André B. Leal[2], Tony J. Dodd[1], and Roderich Groß[1]

[1] Natural Robotics Lab, The University of Sheffield, Sheffield, UK
{y.kaszubowski, t.j.dodd, r.gross}@sheffield.ac.uk
[2] Santa Catarina State University, Joinville-SC, Brazil
andre.leal@udesc.br

Abstract. At present, most of the source code controlling swarm robotic systems is developed in an *ad-hoc* manner. This can make it difficult to maintain these systems and to guarantee that they will accomplish the desired behaviour. Formal approaches can help to solve these issues. However, they do not usually guarantee that the final source code will match the modelled specification. To address this problem, our research explores the application of formal approaches to both synthesise high-level controllers and automatically generate control software for a swarm of robots. The formal approach used in this paper is supervisory control theory. The approach is successfully validated in two experiments using up to 42 Kilobot robots and up to 26 e-puck robots.

1 Introduction

Swarm robotics (SR) studies systems composed of numerous robots that interact and cooperate to achieve certain goals. SR emphasises decentralization of control, limited communication among robots, use of local information, emergence of global behaviour and robustness. Such properties may prove useful in many real-world applications [1].

At present, most of the source code controlling swarm robotic systems is developed in an *ad-hoc* manner, without relying on software engineering methods. This can lead to software that is difficult to maintain. It is also difficult to guarantee that the software will accomplish the desired behaviour.

Formal approaches help to solve or minimise these issues as they require a systematic formalization of the solution. The methods to prove the properties of the system are much more developed to be applied over models expressed by formal approaches than over pure source code. There is a collection of software tools that implement such methods to analyse, validate and even prove properties of systems expressed by formal approaches. Also, the models serve as documentation of the system.

However, even when a project uses formal approaches, it is not guaranteed that the final source code will accomplish its goals. In the context of manufacturing systems, studies have illustrated how control code can be automatically generated using formal approaches [6, 12, 8, 5]. The adaptation of these studies to the SR field could provide a new approach to engineering SR systems and help their transition to real-world applications.

1.1 Formal approaches in swarm robotics

Works have investigated the use of formal approaches in SR. In [9], a group of probabilistic Finite State Machines is used to describe the structure of a swarm of robots at a microscopic level. This work focuses on modelling and analysis rather than control specification and synthesis. The formal specification, synthesis and verification of swarm robotic systems is an active area of research [3, 2].

One of the formal approaches applied to synthesise control logic is Supervisory Control Theory (SCT) [14, 13]. SCT is largely applied in manufacturing systems. In this scenario the decomposition of the model and specifications into several "small" subsystems is applied to solve complex problems.

In [4], the dynamics of a robot team is modelled with Discrete Event Systems (DES) using SCT and aiming at the design of a reconfigurable swarm system, which can handle the situation of robots switching off. However, the focus is on recovery control, the implementation of the control is not considered.

In [16], the authors use Deterministic Finite Automata (DFA) based on SCT to address the task allocation problem in a team of mobile robots, which work together as an automated patrolling/inspection system. In [17], Fuzzy-logic-based utility functions are used to quantify the ability of such robots to perform a task. All these works do not use the full Ramadge and Wonham (RW) framework [14, 13]; instead, they are only partially based on it. As a consequence, much of the software tools and theory development cannot be applied to them.

While some works address the application of formal approaches in SR, there is a lack of work addressing automatic code generation. Moreover, there is a lack of case studies even in the field of manufacturing coordination, which is a major field of application of SCT.

1.2 Contributions

This paper presents the application of SCT to the domain of swarm robotics. It shows how to design and synthesise controllers for the individual robots in a swarm, and how to automatically generate the control software from the formal specification. Two case studies are presented and the same synthesised controller is applied on two target platforms, the Kilobot [15] and the e-puck [10] miniature mobile robotic systems.

The main contributions of this work are (1) to adapt the implementation model of the SCT to the SR field; (2) to apply the SCT using the full RW-framework to SR, from the modelling to the software implementation; (3) to develop a software tool that automatically generates the control software for SR; and (4) to present two case studies applying this software tool and the proposed implementation model.

2 Supervisory control theory

In this section, SCT is overviewed. A language is defined as a set of words over an alphabet Σ, where the alphabet is a set of symbols. The events of a DES are associated with those symbols, and the words formed by those symbols represent sequences of operations. The control objective is to guarantee that at any time only valid words or prefixes of valid words occur. We are interested in a particular class of languages, namely the regular one.

A generator is a formal representation for a regular language used within the SCT framework. It is a quintuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite set of states; Σ is the finite set of symbols related to system events; $\delta : Q \times \Sigma \to Q$ is the partial transition function; q_0 is the initial state, where $q_0 \in Q$; and, Q_m is the set of final, marked as accepting states, where $Q_m \subseteq Q$.

The symbols that represent events are of two types: controllable events (Σ_c) and uncontrollable events (Σ_u), where $\Sigma = \Sigma_c \cup \Sigma_u$ and $\Sigma_c \cap \Sigma_u = \emptyset$. The traditional control operates by receiving stimulus signals from the controlled system and then issuing command signals. Thus, uncontrollable events are stimulus signals (e.g. from a sensor) and controllable events are command signals.

The SCT uses generators to represent free behaviour models and control specifications. The *free behaviour models* abstract each subsystem of each robot to be controlled and represent all of the physical possibilities of the system. We use G_i to represent the i-th free behaviour model. The *specifications* represent the desired behaviours of individual robots. We use E_j to represent the j-th specification model. The generators are synthesised to create supervisors that realise the control logic.

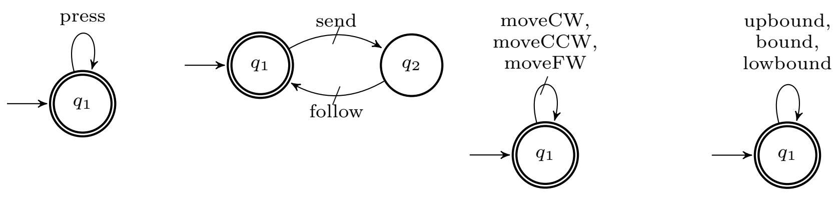
The goal of SCT is to obtain a language, realised by a supervisor, that represents valid sequences of events (in particular, they are *minimally restrictive* [14] and *non-blocking* [18]) respecting the control specifications. The supervisor exerts the control over the robot by disabling controllable events.

3 Modelling swarm robotic behaviours with supervisory control theory

In order to explain the SCT modelling processes, we introduce two didactic case studies, one using two robots to illustrate the robot platforms and the second using a proper swarm. In both case studies all robots use the same control logic and synthesised controller.

3.1 Orbit case study

The orbit strategy [15] is the first case study and it consists of two robots: the static robot and the orbiting robot. The static robot sends an infra-red (IR) message at a regular interval and the orbiting robot uses this message to estimate the distance and then orbit counterclockwise (CCW) around the static robot. When the orbiting robot is inside the boundary interval (which is the initial



(a) G_1 (b) G_2 (c) G_3 (d) G_4

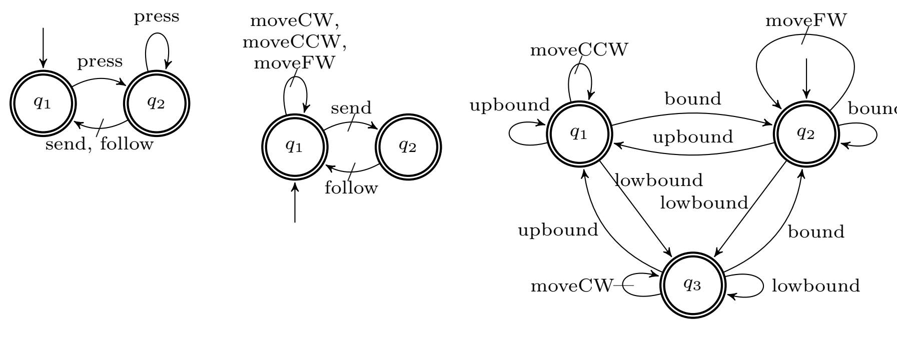
Fig. 1. Free behaviour models for the orbit strategy. Plain arcs represent uncontrollable events. Arcs with a stroke represent controllable events. Each behaviour model, G_i, has its own set of states $Q_i = \{q_1, ..., q_n\}$.

condition), it moves forward. When the distance of both robots is bigger than a threshold (the upper boundary), the orbiting robot turns CCW around the internal wheel/leg axis and approaches the static robot. When the distance of both robots is smaller than a threshold (the lower boundary), the orbiting robot turns clockwise (CW) around the external wheel/leg axis and moves away from the static robot.

Figure 1 shows the free behaviour models for this experiment. The generator G_1 (Figure 1(a)) represents a device to configure the type of the robot (static or orbiting) where the uncontrollable event *press* occurs when the user activates the configuration device. This device can be implemented as an IR signal received from a remote control. The generator G_2 (Figure 1(b)) represents the configuration of the robot: *orbiting* (in state q_1) or *static* (in state q_2). The controllable events *send* and $follow$ start or interrupt the broadcast of the message respectively; the *send* event also stops the robot. This model contains the restriction of both events occuring alternatively.

The generator G_3 (Figure 1(c)) represents the motion capabilities of the robot. The controllable events $moveFW$, $moveCW$ and $moveCCW$ respectively represent the start of the forward movement, the CW turn and the CCW turn. The robot executes that movement indefinitely. The generator G_4 (Figure 1(d)) represents the boundary sensor; the uncontrollable events *upbound*, *bound* and *lowbound* are triggered when the orbiting robot is respectively too far, inside the boundary, or too near. As the distance estimation by IR message is not precise, the robot can receive the uncontrollable events in any order and at any time. Those events are continuously triggered after a sampling cycle of $200ms$.

Figure 2 shows the specifications of the orbit strategy. Figure 2(a) shows the specification of the user interaction to configure the robot type. Figure 2(b) shows the specification that enables the movement only for the orbiting robot. Figure 2(c) implements the main rule of the strategy, as previously described. States q_1, q_2 and q_3 (in E_3) specify the motion of the orbiting robot when it is respectively too far, inside the boundary or too near. As the initial state is q_2, it is considered that the robot starts inside the boundary.



(a) E_1 (b) E_2 (c) E_3

Fig. 2. Specification for the orbit strategy. Each specification, E_j, has its own set of states $Q_j = \{q_1, ..., q_m\}$.

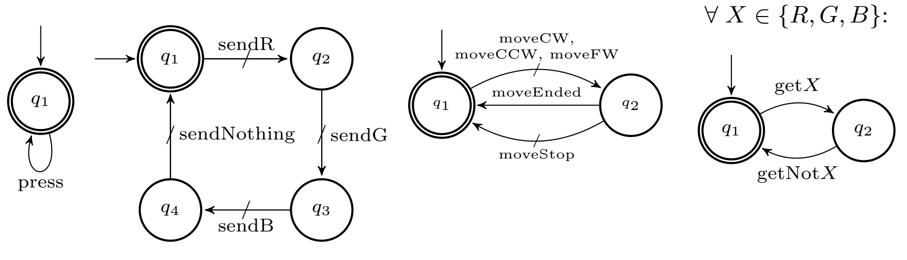
3.2 Segregation case study

The second case study addresses a segregation strategy. Each robot is configured as a leader or as a follower at the beginning of the trial. There are three types of leaders, which we refer to as the red, the green and the blue. These leaders are initially spread in the arena. They broadcast by IR a message, at fixed intervals of time, containing their type/colour. The follower robots that are in the signal field of only one type of leader start to belong to that leader. If a robot receives the signal from two or more different types of leaders, this robot starts to move at random until it finds a position where it receives signals from only one type of leader or no signal.

Figure 3 shows the free behaviour models for this experiment. The free behaviour G_1 (Figure 3(a)) represents a device to configure the type of the robot as in the previous experiment. The free behaviour G_2 (Figure 3(b)) represents the message transmission when a robot is a leader and no transmission when it is a follower. Initially, each robot is a follower robot (state q_1 in G_2). The controllable events *sendR*, *sendG* and *sendB* start the broadcast of the messages red, green and blue, respectively. The controllable event *sendNothing* stops the broadcast.

The free behaviour model G_3 (Figure 3(c)) defines the motion behaviour. The controllable events *moveFW*, *moveCW* and *moveCCW* start the forward movement, the clockwise turn and counterclockwise turn of the robot for a random period, respectively. An uncontrollable event *moveEnded* is generated at the end of this period. The controllable event *move Stop* forces the end of the movement. Initially, the robots do not move.

The free behaviour models G_4, G_5 and G_6 (Figure 3(d)) represent the receiving of a message from the leaders red, green and blue respectively. The uncontrollable event $getX,\ X \in \{R, G, B\}$ occurs if the robot starts to receive a message from the corresponding type of leader; $getNotX,\ X \in \{R, G, B\}$ occurs



(a) G_1 (b) G_2 (c) G_3 (d) $G_4, ..., G_6$

Fig. 3. Free behaviour models for the segregation strategy. Final states Q_m are indicated by double lines.

when the robot stops receiving messages from the corresponding type of leader. A sampling cycle of $200ms$ is used to check if the robot is receiving the message.

Figure 4 illustrates the specifications to implement the robot type configuration and the segregation strategy. Figure 4(a) shows the specification E_1, which relates the user input with the change of robot type. Each time the user gives an input signal (the *press* event) the event to change the robot type is enabled. The combination with the model G_2 (Figure 3(b)) will guarantee the sequential order of types: follower, red leader, green leader and blue leader. This allows the user to set a robot as one of the three types of leaders or as a follower robot. Figure 4(b) shows the specification E_2 that enables the movement only for the followers and message broadcasts only for the leaders.

Figure 4(c) represents the main rule of the strategy, the specification E_3. When in state q_1 (not receiving any signal) or state q_2 (receiving signals from only one type of leader) the robot is forbidden to move and only the stop event (*moveStop*) is enabled. In state q_3 (receiving signals from two types of leaders) or state q_4 (receiving signals from three types of leaders) the move events are enabled and the stop event (*moveStop*) is disabled. The specification E_3 can be seen as a counter, in state q_1 there is no signal being received, in state q_2 there are signals for one type of leader, in state q_3 there are signals for two types of leaders, and in state q_4 there are signals for all three types of leaders.

The follower robot alternates its movement between the three different modes (forward, CW turn and CCW turn) when receiving signals from more than one type of leader. Each time when the event *moveEnded* occurs, it changes to state q_1 in G_3 where the events $moveFW$, $moveCW$ and $moveCCW$ are all enabled. As the choice for controllable events is at random, the robot will move randomly.

4 Supervisor synthesis

The initial approach to synthesise a supervisor is the monolithic approach, where for all free behaviour and specification models only one supervisor S is obtained. The parallel composition (represented by $||$) of two generators G_a and G_b is

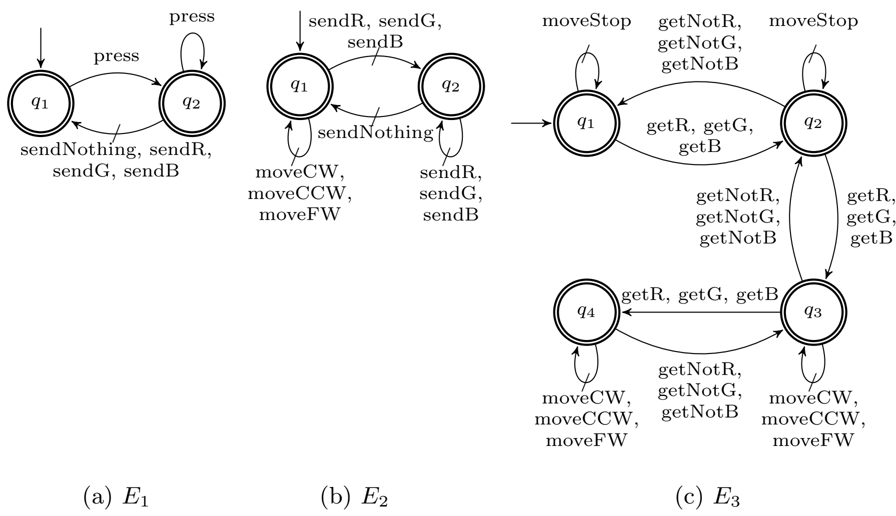


Fig. 4. Specification for the segregation strategy.

defined as follows:

$$G_a||G_b = (Q_a \times Q_b, \Sigma_a \cup \Sigma_b, \delta_{a||b}, (q_{0_a}, q_{0_b}), Q_{m_a} \times Q_{m_b}), \tag{1}$$

where

$$\delta_{a||b}((q_a, q_b), e) = \begin{cases} (\delta_a(q_a, e), \delta_b(q_b, e)) & \text{if } \ \delta_a(q_a, e)! \wedge \delta_b(q_b, e)! \\ (\delta_a(q_a, e), q_b) & \text{if } \ \delta_a(q_a, e)! \quad \text{only} \\ (q_a, \delta_b(q_b, e)) & \text{if } \ \delta_b(q_b, e)! \quad \text{only} \\ \quad \text{undefined} & \text{otherwise,} \end{cases} \tag{2}$$

and $\delta(a)!$ means that function δ is defined for a.

A generator is accessible if all its states are reachable from the initial state q_0. A generator is coaccessible if all its states can reach at least one final state. The $Trim(G)$ operation removes all the non-accessible and non-coaccessible states from G. Let us consider the previous orbit case study with ($n = 4$) free behaviour models and ($m = 3$) specification models. The first step is the parallel composition for all free behaviour models and specifications. That is,

$$G = G_1|| \cdots ||G_4, \tag{3}$$

$$E = E_1||E_2||E_3. \tag{4}$$

To obtain a supervisor, S, which controls the free behaviour G, it is necessary to calculate the target language K, which is defined as the following parallel composition,

$$K = G||E. \tag{5}$$

As a result of the parallel composition, each state q_x^K in K is mapped on a state q_y^G in G. This mapping is not necessarily injective. One says that q_x^K

is a composed state $(q_y^G, .)$. An event e is enabled in a state q if $\delta(q, e)!$. If an event e is enabled in q_y^G but not enabled in $q_x^K = (q_y^G, .)$, it means that the event is physically possible to occur, but the control specification denied it to occur. In this case q_x^K is called a bad state and if it is reached, the undesired event e cannot be disabled by the supervisor. Thus, q_x^K and any uncontrollable path to q_x^K must be removed.

To obtain a supervisor S where its supremal controllable sublanguage is $L_m(S/G)$, the iterative removal of bad states and the $Trim$ operation is performed by the $SupC$ operator as shown in Equation 6. This operator removes all bad states and each state that leads to the bad state through an uncontrollable path, that is, each state $q_a : \exists s \in \Sigma_u^+ : \delta(q_a, s) = q_{bad}$. Finally, K is modified by the $Trim$ component and the iterative process restarts until it does not modify K anymore.

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

The obtained supervisor, S, represents the control logic according to the designed model and can be used to implement the physical controller. However, the number of states and transitions grows exponentially by the parallel composition, which in some cases may not be feasible. To solve this problem, another approach, called modular supervisors, has been proposed by [18] and it was extended to local modular supervisors by [11].

4.1 Local modular supervisors

In this approach one supervisor is created for each control specification and only the free behaviour models that are affected by the control specification are composed in the calculus of each modular supervisor. Thus, each specification has its own local free behaviour model G_j^{loc}. The G_j^{loc} is the parallel composition of each free behaviour G_i which has at least one event in common with E_j. The local free behaviours for the orbit case are:

$$G_1^{loc} = G_1||G_2,\ G_2^{loc} = G_2||G_3,\ G_3^{loc} = G_3||G_4. \tag{7}$$

For the segregation case, the local free behaviours are:

$$G_1^{loc} = G_1||G_2,\ G_2^{loc} = G_2||G_3,\ G_3^{loc} = G_3||...||G_6. \tag{8}$$

The supervisor for both cases are obtained as,

$$\forall x \in \{1, 2, 3\} : K_x = G_x^{loc}||E_x, \tag{9}$$

$$S_x\ :\ L_m(S_x/G_x^{loc}) = SupC(G_x^{loc}, K_x). \tag{10}$$

In all supervisors for both cases all target languages K_x are already controllable, that is, $K_x = S_x$. This characteristic implies that the case studies are relatively simple. However, the focus of this work is to introduce the application of SCT to swarm robotics. The case studies were chosen to help illustrate the theory.

Applying the local modular approach requires that there is no conflict between supervisors (which could result in deadlocks). To test this, the monolithic system is built and compared with the composition of the local modular supervisors. In the case studies considered here there are no conflicts. If conflicts are present, they can be resolved by replacing the individual supervisors in conflict with a composite supervisor for all the conflicting specifications.

5 Implementation of supervisory control in SR

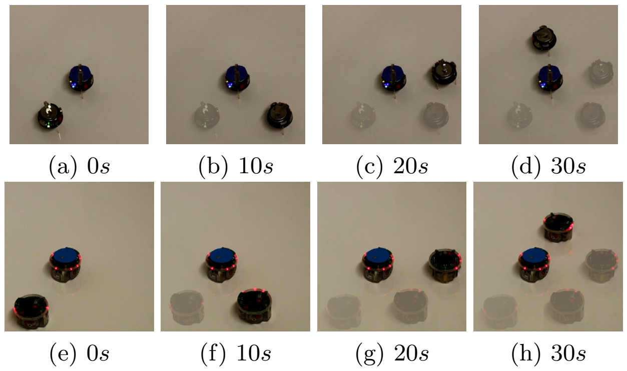
Our implementation is based on the SCT architecture proposed by [12]. However, we use the complete local modular supervisors instead of the reduced ones with the product system applied by [12]. The difference between these are small and are not detailed here. Those supervisors that are assigned to disable the controllable events are stored in the generated data structure proposed by [8].

The core of the controller is the *automata player*, which accesses the generated data to control the flow of the controller logic and it is in charge of evolving the generators. To do so, it stores the current state of each generator. An arbitrary number of supervisors can run in parallel in the same control structure managed by the automata player. Besides the generated data, the automata player evolves according to the occurrence of events defined by the operational procedures.

The *operational procedures* are a low-level interface between the supervisors and the real system that works by generating the control system output and reading the input [12]. As the architecture was designed to be applied mainly in manufacturing coordination the operational procedures are mostly applied to translate the events to the high or low signal in the output pins from Programmable Control Logic devices or to translate its pin signals to events. However, as shown in the following, the operational procedures layer is able to perform more complex tasks.

We associate each event to user defined callback functions to perform the operational procedures as proposed by [8]. We extend the generation tool called Nadzoru [8] to support the e-puck and Kilobot platforms and change the method to insert the operational procedure code. Instead of including the code inside the Nadzoru, now the Nadzoru allows the developer to specify external callback functions. Also, the control of the main loop function is delegated to the user, which must call an update function every cycle. This allows more flexibility to the developer.

The *user code* implements the operational procedures. The developer registers in the initialisation of the code a callback function for each event; this function is called every time that the event happens. Furthermore, the developer must register one callback function for each uncontrollable event to check whether the event occurs. All our implementations, models and the Nadzoru tool can be found in [7].


(a) 0s (b) 10s (c) 20s (d) 30s

(e) 0s (f) 10s (g) 20s (h) 30s

Fig. 5. Sequence of (superimposed) snapshots taken from one of the trials where two Kilobots (a-d) and two e-pucks (e-h) perform the orbit task.

6 Experiment

This section presents two experiments that validate our application of SCT to swarm robotics. In both experiments the local modular approach is used. Two types of platforms are considered, and for both the same formal approach, free behaviours and specification models are used. This is one advantage of the approach. However, it requires that all devices must have the capability to perform the desired task. Thus, the task is modelled once for all devices and the device differences are abstracted and compensated in the operational procedures layer (user code).

Both experiments take place in a 1.20m $\times$ 0.90m two-dimensional arena. The configuration of the orbit experiment consists of two robots. The static robot is placed in the centre of the arena and the orbiting robot is placed inside (or near to) a configured boundary. Ten trials are performed for each type of robot. Each trial is limited to 300s. The experiment evaluates the match of the modelled specifications with the synthesised control logic. We observed that the robots behaved according to the specifications. Figure 5 shows snapshots taken from the experimental trials with the Kilobots and e-pucks.

The configuration of the segregation experiment consists of a group of 39 follower robots in the Kilobot experiment and 20 follower robots in the e-puck experiment. These robots are distributed on a grid. Three leader robots are placed inside the grid in the Kilobot case; in the e-puck case configurations with three pairs of leaders are applied. Ten trials are performed. Each trial runs for 300s or until the robots are segregated, whichever occurs first. The robots are considered to be segregated if they all receive a signal of only one leader or no signal at all (for details, see Figure 6). The experiment evaluates the match of the modelled specifications with the synthesised control logic. We observed that in all trials the robots behaved according to the specifications. Figure 6 shows snapshots taken from two trials.

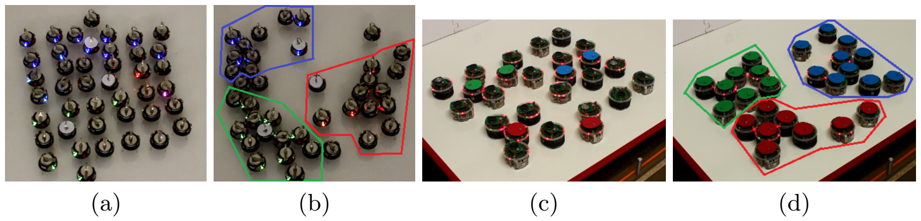


Fig. 6. Snapshots from a segregation trial with Kilobots: (a) initial grid formation with three leaders, marked with tags; (b) result after segregation occurred. Trial with e-pucks: (c) initial grid formation with three pairs of leaders marked with tags; (d) result after segregation occurred, tags were added after the experiments for visualisation (based on the robots states as indicated by their Light-Emitting Diodes (LEDs)).

Video recordings from the experiments and additional resources (models, the Nadzoru tool, the source code used) can be found in [7].

7 Conclusions

This paper presented the application of Supervisory Control Theory (SCT) to swarm robotics and validated it through two case studies. First, the basic concepts were presented in a swarm robotics context. Second, an implementation for SCT, based on the architecture proposed by [12], was applied to swarm robotics. Third, the Nadzoru tool was extended to support code generation for two swarm robotics platforms, Kilobot and e-puck. Finally, the synthesised control logic was validated in experiments with these two platforms using the same formally synthesised control logic.

The use of formal approaches brings several advantages for the design of swarm robotic systems. Given a description of a system's capabilities and a set of specifications for the desirable behaviour of the individual robots, the control logic can be obtained. Moreover, the control software can be generated automatically. The control logic models can even be used for code generation in different types of platforms. In this paper, we illustrated this by two case studies with Kilobot and e-puck robot swarms. The only code that had to be manually written are the operational procedures, which is only a small fraction of the amount of code generated when the whole logic was implemented in an *ad-hoc* manner (see additional materials in [7]). Moreover, the use of operational procedures resulted in a more intuitive code that links events with framework functions.

In the future we will investigate how to prove properties of swarm robotic systems modelled by SCT. Also, we will consider more complex case studies.

Acknowledgements Y.K. Lopes acknowledges support by CAPES - Brazil (grant number: 0462/12-8).

References

1. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence 7(1), 1–41 (2013)
2. Brambilla, M., Pinciroli, C., Birattari, M., Dorigo, M.: Property-driven design for swarm robotics. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. pp. 139–146 (2012)
3. Dixon, C., Winfield, A., Fisher, M.: Towards temporal verification of emergent behaviours in swarm robotic systems. In: Groß, R., Alboul, L., Melhuish, C., Witkowski, M., Prescott, T.J., Penders, J. (eds.) TAROS. Lecture Notes in Computer Science, vol. 6856, pp. 336–347. Springer (2011)
4. Gordon-Spears, D., Kiriakidis, K.: Reconfigurable robot teams: modeling and supervisory control. IEEE Transactions on Control Systems Technology 12(5), 763–769 (2004)
5. Leal, A.B., Cruz, D.L.L., Hounsell, M.S.: PLC-based implementation of local modular supervisory control for manufacturing systems. In: Aziz, F.A. (ed.) Manufacturing System, pp. 159–182. InTech (2012)
6. Liu, J., Darabi, H.: Ladder logic implementation of Ramdge-Wonham supervisory controller. Proc. of the WODES pp. 383–392 (2002)
7. Lopes, Y.K., Leal, A.B., Dodd, T.J., Groß, R.: Online supplementary material (2014), http://naturalrobotics.group.shef.ac.uk/supp/2014-001/
8. Lopes, Y.K., Leal, A.B., Rosso, R.S.U., Harbs, E.: Local modular supervisory implementation in micro-controller. In: Proc. of the 9th International Conference of Modeling, Optimization and Simulation (MOSIM 2012). vol. 9 (2012)
9. Martinoli, A., Easton, K., Agassounon, W.: Modeling swarm robotic systems: A case study in collaborative distributed manipulation. Int. Journal of Robotics Research 23(4-5), 415–436 (2004)
10. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: 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 (2009)
11. Queiroz, M.H., Cury, J.E.R.: Modular control of composed system. In: Proceedings of the American Control Conference, Chicago. pp. 4051–4055 (2000)
12. Queiroz, M.H., Cury, J.E.R.: Synthesis and implementation of local modular supervisory control for a manufacturing cell. In: Proceedings of International Workshop on Discrete Event Systems (WODES). pp. 103–110 (2002)
13. Ramadge, P.J., Wonham, W.: The control of discrete event systems. Proceedings of the IEEE 77(1), 81–98 (1989)
14. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event process. SIAM J. Control and Optimization 25(1), 206–230 (1987)
15. Rubenstein, M., Ahler, C., Nagpal, R.: Kilobot: A low cost scalable robot system for collective behaviors. In: Proccedings of ICRA 2012. pp. 3293–3298. IEEE (2012)
16. Tsalatsanis, A., Yalcin, A., Valavanis, K.: Optimized task allocation in cooperative robot teams. In: Proc. of the 17th Mediterranean Conference on Control and Automation (MED'09). pp. 270–275 (2009)
17. Tsalatsanis, A., Yalcin, A., Valavanis, K.P.: Dynamic task allocation in cooperative robot teams. Robotica 30(5), 721–730 (2012)
18. Wonham, W., Ramadge, P.J.: Modular supervisory control of discrete event system. Mathematics of control, signals and systems 1(1), 13–30 (1988)