Probabilistic Supervisory Control Theory (pSCT) Applied to Swarm Robotics

Yuri Kaszubowski Lopes
The University of Sheffield, Department of Automatic Control and Systems Engineering
Sheffield, UK
y.kaszubowski@sheffield.ac.uk

Stefan M. Trenkwalder
The University of Sheffield, Department of Automatic Control and Systems Engineering
Sheffield, UK
s.trenkwalder@sheffield.ac.uk

André B. Leal
Santa Catarina State University, Department of Electrical Engineering
Joinville, SC, Brazil
andre.leal@udesc.br

Tony J. Dodd
The University of Sheffield, Department of Automatic Control and Systems Engineering
Sheffield, UK
t.j.dodd@sheffield.ac.uk

Roderich Groß
The University of Sheffield, Department of Automatic Control and Systems Engineering
Sheffield, UK
r.gross@sheffield.ac.uk

ABSTRACT

Swarm robotics studies large groups of robots that work together to accomplish common tasks. Much of the used source code is developed in an ad-hoc manner, meaning that the correctness of the controller is not always verifiable. In previous work, supervisory control theory (SCT) and associated design tools have been used to address this problem. Given a formal description of the swarm's agents capabilities and their desired behaviour, the control source code can be automatically generated. However, regular SCT cannot model probabilistic controllers (supervisors). In this paper, we propose a probabilistic supervisory control theory (pSCT) framework. It applies prior work on probabilistic generators in a way that allows controllers to be decomposed into multiple local modular supervisors. Local modular supervisors take advantage of the modularity of formal specifications to reduce the size required to store the control logic. To validate the pSCT framework, we model a distributed swarm robotic version of the graph colouring problem and automatically generate the control source code for the Kilobot swarm robotics platform. We report the results of systematic experiments with swarms of 25 and 100 physical robots.

CCS Concepts

•**Computing methodologies** → **Multi-agent systems;**

Appears in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

Keywords

Swarm robotics, Supervisory control theory, Probabilistic generator, Kilobot

1. INTRODUCTION

Swarm robotics studies large groups of robots and how they can solve tasks by using simple rules and interactions [1, 2]. One of the challenges in swarm robotics is the design and analysis of those rules and interactions, as they can cause unpredictable behaviour. Many automated and formal methods were considered. Examples are the use of algebraic graph theory to analyse stability properties of flocking agents [3], the verification of swarm robotics properties with temporal logic [4], a top-down design method that uses prescriptive modelling and model checking [5], and methods for automatically generating probabilistic finite state machines [6].

In this paper, we focus on supervisory control theory (SCT) [7, 8], which has been used to formally control discrete event systems. SCT has been mainly studied in the context of manufacturing, where supervisors have been applied to programmable logic controllers (PLC) [9].

In SCT, languages are used to synthesise the control logic using a formal approach. The class of regular languages—expressed as generators—is widely used due to its simplicity. Generators are similar to automata; the difference is that automata verify if a word, given as an input, belongs to a language whereas generators produce words that belong to the language. In SCT, some generators—called free behaviour models—represent the model of the system whereas others represent the control specification. All generators can be combined to obtain a monolithic generator called supervisor. The supervisor restricts the behaviours of the system to those that do not violate the control specification.

Systems are often a composition of many subsystems [10]. The composition of these subsystems may result in a rapid

increase in the number of states, and render the monolithic supervisor approach infeasible, especially when considering swarms of robots with limited computational resources. To alleviate this problem, supervisors can be divided into modules [11, 12, 10, 13]. This modularity allows the use of multiple supervisors that are smaller—regarding the number of states and transitions—than the equivalent monolithic supervisor.

The traditional SCT is unable to formulate probabilistic controllers at the supervisor level. Probabilities, if needed, must be considered on a lower layer called operational procedures [13], which is responsible for establishing the link between the abstract events and the physical actuators and sensors.

To formulate probabilistic controllers at the supervisor level, we propose a probabilistic SCT (pSCT) framework in the context of swarm robotics. The framework combines the concept of probabilistic generators [14, 15] with support for marked states[1] and the synthesis of local modular supervisors [12, 10, 13]. The latter results in supervisors with more compact memory representation.

Probabilistic generators differ from probabilistic automata. Probabilistic automata are concerned with the uncertainty of the system's state, which is defined by a stochastic vector. In a particular state, each event may have transitions to multiple states (with associated probabilities). Note that this is one approach to represent actuation uncertainty. In [16], actuation uncertainty is represented by probabilistic computation tree logic in the context of a stochastic motion planning task.

Probabilistic generators, on the other hand, are concerned with the uncertainty of events occurring in the system's state, which is assumed to be known. Each event will result in a transition to a single state.

In the context of SCT, the events can be uncontrollable or controllable. Uncontrollable events are usually related with the controller input, for example, from the sensors of a robot. Controllable events are usually related with the controller output (i.e., the system's input), in other words, they can relate to a choosable action. The problem of selecting one of multiple controllable events—associated with the transitions in the current state—is referred to as the choice problem [17]. We show how the use of probabilistic generators in conjunction with the proposed pSCT represents a systematic and formally explicit solution for the choice problem.

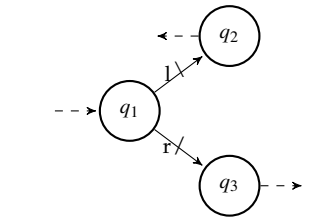
To didactically introduce pSCT, we make use of a case study from the domain of swarm robotics. We apply the pSCT framework to control a swarm of real robots that distributively solve the graph colouring problem [18]. pSCT automatically generates the controller's code, using the software tool Nadzoru [19], and applies it on physical swarms of 25 and 100 Kilobot robots [20].

The paper is structured as follows. The choice problem and its solution by generalising SCT to pSCT is presented in Section 2. Sections 3, 4, and 5 present the modelling, the synthesis, and the implementation of probabilistic generators, respectively. The experimental validation on physical robots is presented in Section 6. The paper is concluded in Section 7.

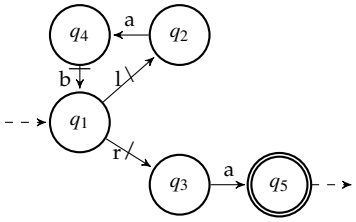
2. SCT AND PSCT

A generator G is defined as:

[1]Marked states are states that represents a goal for the system (e.g., they could correspond to the end of a task).



(a)



(b)

Figure 1: (a) A choice problem between two enabled controllable events from state q_1. (b) A choice problem with livelock. If the same transition (l) is repeatedly chosen from q_1, a livelock can occur.

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

where Q is a finite set of states. Σ is the finite set of events. Events can be either controllable (Σ_c) or uncontrollable (Σ_u), in other words, $\Sigma = \Sigma_c \cup \Sigma_u$ and $\Sigma_c \cap \Sigma_u = \emptyset$. Controllable events represent commands issued by the controller, uncontrollable events represent feedback signals. $\delta : Q \times \Sigma \to Q$ is the partial transition function. q_0 is the initial state, where $q_0 \in Q$. Q_m is the set of marked states where $Q_m \subseteq Q$.

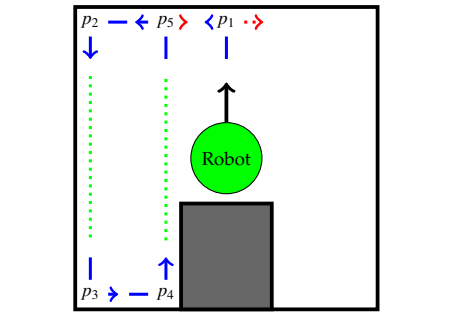
2.1 Limitations of traditional SCT

Ideally, at the implementation level, no state should have more than one enabled controllable event. In this way, for any input sequence, there is a single response. In reality, this is not always the case, as the specifications may not restrict all the possibilities.

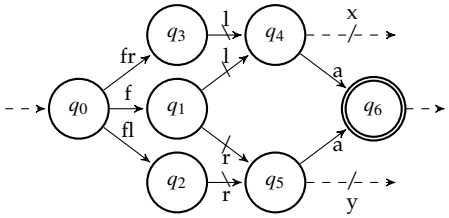
For example, let us consider the supervisor shown in Figure 1(a). We represent generators using graphs where plain arcs represent uncontrollable events, and arcs with a stroke represent controllable events. Some states of the supervisors are omitted, the transition from/to the omitted part of the supervisor are represented by dashed lines. In state q_1 two controllable events, l and r, are enabled. Such case occurs if a robot, which moves forward, encounters an obstacle (e.g., a wall) and can either avoid it by moving to the left (event l) or the right (event r). From the perspective of the supervisor both control responses are permissible, as respecting the specification. The choice problem occurs at the implementation level. When in a particular state two or more controllable events are enabled, the controller's implementation has to choose which controllable event will be generated.

If the choice strategy is deterministic it can result in the controller being trapped in a subset of non-marked states. This is referred to as a livelock.[2]

[2]If a controller is trapped in a single non-marked state this is called a deadlock. Deadlocks can be prevented during the synthesis (see Section 4).



(a)



(b)

Figure 2: (a) A choice problem without livelocks. Some states are omitted. (b) An example implementation that underperforms due to the choice problem.

The supervisor shown in Figure 1(b) is deadlock free, as each state has at least one path that reaches a marked state (q_5 in this case). Let us assume that the implementation always chooses the first option in state q_1—the transition triggered by event l. The result is equivalent to removing the other transitions triggered by controllable events. In this case a livelock occurs as the controller will be trapped in states q_1, q_2, and q_4 executing the events l, a, b in this order indefinitely.

Even when there is no livelock, the strategy of selecting the same transition in a state can also lead to problems. Consider a robot that starts moving upwards from the central position of the arena, as shown in Figure 2(a). Once it has reached position $p_1, p_2, \cdots$, or p_5, its supervisor is assumed to be in state q_0 (see Figure 2(b)). The uncontrollable events f, fl, and fr represent the sensing of an obstacle "*in front*", "*in front and on the left*", and "*in front and on the right*", respectively. Controllable events l and r move the robot to the left or the right, respectively. In the position p_1, there is an obstacle in the front, the supervisor thus reaches state q_1 (Figure 2(b)). If the controller always chooses l in state q_1, this will not cause a livelock, because it is possible for the generator to reach a marked state (q_6). In the positions p_2, p_3, and p_4, there are obstacles (walls) in the front and on the right side; accordingly, the supervisor reaches state q_3 (Figure 2(a)), which only allows to turn left. Similarly to position p_1, in position p_5, the supervisor reaches state q_1. As the controller implementation triggers only the controllable event l in state q_1, the robot will never turn right, even though the event related to turning right, r, is enabled in state q_1. As a result, the robot is only exploring half of the arena. This behaviour is solely caused by an inadequate implementation and is not restricted by the supervisor.

Therefore, if multiple controllable events are enabled, the controller should not always have to choose the same event. It can be shown that if the choices are made deterministically, the robot may repeat the same sequence of actions indefinitely. A better, and more common, approach is to choose one of the enabled controllable events randomly. This will avoid livelocks and unspecified restrictions, as eventually all the controllable transitions will be chosen.

A uniformly random selection among the enabled controllable events is only a basic solution. However, it may be desirable that some events occur more often than others. For example, moving forward may be required to be more prominent than the turning movements. This was the case for the random movement used in the segregation strategy presented in [21]. While the events move forward, turn left, and turn right occurred, statistically, in the same proportion, at the implementation level (the operational procedures), the forward movement was performed for a longer time than the turning movements. Consequently, the differences were not part of the formal modelling and specifications.

2.2 Probabilistic generators and pSCT

Probabilistic generators are able to formally represent different likelihoods of controllable events. Different definitions for probabilistic generators have been proposed. In [15] probabilistic generators are defined as:

$$G^{p'} = \{Q, \Sigma, \delta, q_0, p\}, \tag{2}$$

where Q, Σ, δ, and q_0, are defined as in Equation 1. The probability of an event occurring in a particular state is:

$$p : Q \times \Sigma \to [0, 1], \tag{3}$$

where the sum of the probabilities for each state is limited to 1, as:

$$\forall q \in Q, \sum_{e \in \Sigma} p(q, e) \leq 1. \tag{4}$$

If each state q in a generator G holds $\sum_{e \in \Sigma} p(q, e) = 1$, then G is non-terminating. Otherwise, G is terminating—when no event e occurs, the generator stops.

A different definition for probabilistic generators, given by [14], includes marked states, Q_m:

$$G^p = \{Q, \Sigma, \delta, q_0, Q_m, p\}. \tag{5}$$

This definition has been applied to the synthesis of a supervisor defined by a single free behaviour model and a single specification [14].

The probabilistic supervisory control theory (pSCT) defines probabilistic generators as in Equation 5 but drops the restriction of Equation 4. In pSCT there are no terminating states. We use the p values to weigh the occurrence of events in each state separately for each generator. When combining multiple generators, as will be shown later, their control logic is only guaranteed to be preserved, if a normalisation of weights is applied once, rather than for each individual generator. For the sake of simplicity, we refer to the weights as probabilities.

Non-probabilistic generators used by the traditional SCT can be expressed as probabilistic generators in our pSCT framework. Figure 3 shows the probabilistic version of the generator presented in Figure 2(b). The label of each transition is represented by $e_c : p$, where e_c is a controllable event and $0 \leq p \leq 1$ is the probability of the transition. For this particular example, we assume that the controllable events in each state should be chosen with equal probabilities, which is a common workaround at the implementation level of the traditional SCT.

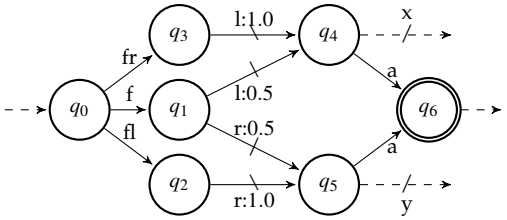


Figure 3: Example of a probabilistic generator derived from a non-probabilistic generator.

2.3 Operations for the synthesis of probabilistic supervisors

This section defines operations needed to synthesise supervisors based on the proposed probabilistic generators. The goal of the synthesis is to obtain a supervisor that enables only those controllable events that cannot cause a violation of the specification. More precisely, the supervisor must realise a language that is controllable (and hence non-blocking) and minimally restrictive. The use of such operations will be detailed later using a case study.

2.3.1 Normalisation

The normalisation operation of a generator G^p, $Norm(G^p)$, guarantees that in each state the sum of all probabilities of the transitions related to the controllable events is equal to 1. The normalised probability $p_n(q,e_{c_x})$, with $e_{c_x} \in \Sigma_c$, is given by:

$$p_n(q,e_{c_x}) = \begin{cases} p(q,e_{c_x}) \,/ \sum\limits_{e_c \in \Sigma_c} p(q,e_c) & \text{if } \sum\limits_{e_c \in \Sigma_c} p(q,e_c) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

2.3.2 Synchronisation

The synchronous composition (represented by $\cdot||\cdot$) of two probabilistic generators G_a^p and G_b^p with alphabet Σ_i, $i \in \{a,b\}$ is defined as:

$$G_a^p||G_b^p = (Q_a \times Q_b, \Sigma_a \cup \Sigma_b, \delta_{a||b}, (q_{0_a}, q_{0_b}), Q_{m_a} \times Q_{m_b}, p_{a||b}), \tag{7}$$

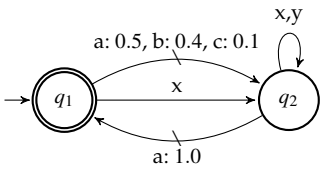
where

$$\delta_{a||b}((q_a,q_b),e) = \begin{cases} (\delta_a(q_a,e),\delta_b(q_b,e)) & \text{if } \delta_a(q_a,e)! \wedge \delta_b(q_b,e)! \\ (\delta_a(q_a,e),q_b) & \text{if } \delta_a(q_a,e)! \wedge e \notin \Sigma_b \\ (q_a,\delta_b(q_b,e)) & \text{if } \delta_b(q_b,e)! \wedge e \notin \Sigma_a \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{8}$$

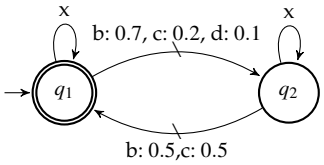
with $\delta(x,y)!$ meaning that δ is defined for an input (x,y). The probability of transitions triggered by controllable events is:

$$p_{a||b}((q_a,q_b),e_c) = \begin{cases} p_a(q_a,e_c) \times p_b(q_b,e_c) & \text{if } \delta_a(q_a,e_c)! \wedge \delta_b(q_b,e_c)! \\ p_a(q_a,e_c) & \text{if } \delta_a(q_a,e_c)! \wedge e_c \notin \Sigma_b \\ p_b(q_b,e_c) & \text{if } \delta_b(q_b,e_c)! \wedge e_c \notin \Sigma_a \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

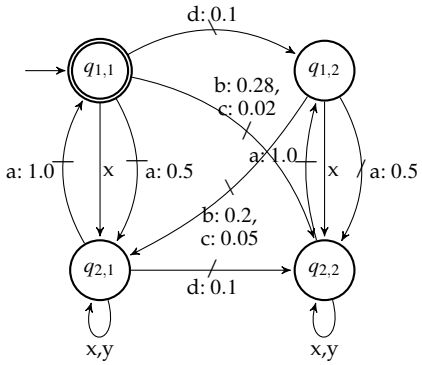
Figure 4 shows an example of the synchronous composition of two probabilistic generators, G_1^p and G_2^p. Note that the resulting generator (Figure 4(c)) is not necessarily normalised.



(a) G_1^p, $\Sigma_u^1 = \{x,y\}$, $\Sigma_c^1 = \{a,b,c\}$



(b) G_2^p, $\Sigma_u^2 = \{x\}$, $\Sigma_c^2 = \{b,c,d\}$



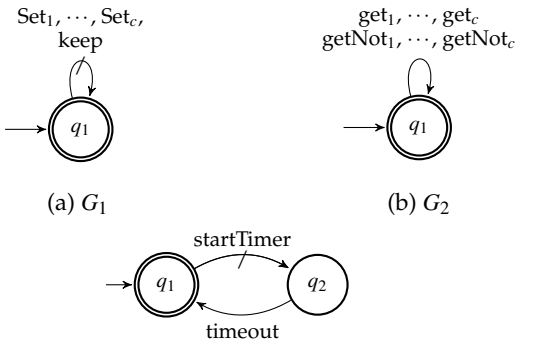
(c) $G_{1,2}^p$, $\Sigma_u^{(1,2)} = \{x,y\}$, $\Sigma_c^{(1,2)} = \{a,b,c,d\}$

Figure 4: Example of the synchronous composition of two probabilistic generators. (a) G_1^p and (b) G_2^p, respectively are combined to form (c) $G_{1,2}^p$.

3. GRAPH COLOURING CASE STUDY

Our case study derives from the classical graph colouring problem [18]. The system comprises an arbitrary number of robots, r, and an arbitrary number of colours that can be chosen, c. Each robot can be seen as a node in a graph. Two robots R_a and R_b share an edge and are therefore neighbours if their distance is smaller than a threshold d. The goal is to assign a colour to each robot in such a way that any pair of neighbours do not have the same colour while the number of different colours used by the entire swarm should be minimal. A practical application of the graph colouring problem in swarm robotics is to assign locally unique identification numbers to robots when the overall size of the swarm is not a priori known.

In the following, we present a heuristic strategy for addressing the graph colouring problem. The free behaviour models (i.e., models of the robot's abilities) for the graph colouring strategy are illustrated in Figure 5. Based on the availability of c colours, free behaviour model G_1 defines the controllable events set_x with $x \in \{1,\cdots,c\}$. set_x sets the colour of the robot to x and starts to broadcast a message informing the neighbour robots of its decision. Controllable event *keep* preserves the previous selection. Free behaviour model G_2 defines the uncontrollable events get_x and $getNot_x$ with



(a) G_1 (b) G_2

(c) G_3

Figure 5: Free behaviour models for the graph colouring case study. (a) The robot's ability to assume one of c colours; (b) the robot's ability to receive messages informing it of the colour of nearby robots; (c) robot's internal timer.

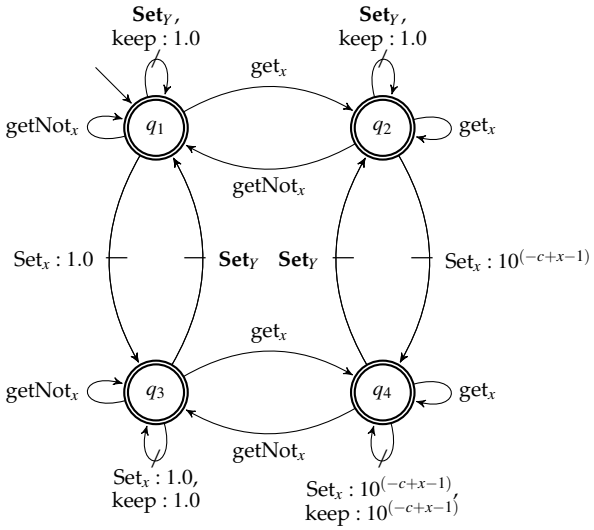
$x \in \{1, \cdots, c\}$. get_x occurs when at least one message over a time interval of 2 s has been received stating that a neighbour has chosen the colour x, otherwise $getNot_x$ occurs. Free behaviour model G_3 represents a timer that is triggered every 4 s.

Figure 6 shows the control specification realising the graph colouring strategy. Specifications $E_{(1,x)}$ (Figure 6(a)), with $x \in \{1, \cdots, c\}$, set a lower probability for selecting a colour x, if it is known that a neighbour already had selected it—in states q_2 and q_4. The specifications also set a lower probability of keeping the current colour (event *keep*) if a neighbour has the same colour already selected—in state q_4.

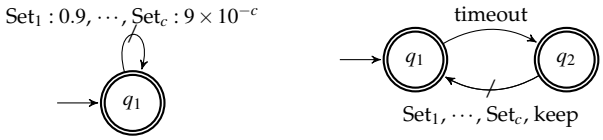
Specification E_2 (Figure 6(b)) defines the default probability of a colour being selected.[3] The x-th colour, with $x \in \{1, \cdots, c\}$, has probability 9×10^{-x} of being selected. For example, Set$_1$ has probability 0.9, Set$_2$ has probability 0.09, Set$_3$ has probability 0.009, and so on.

Specification E_3 (Figure 6(c)) establishes a waiting period for any change of colour to happen.

$\forall\, x \in \{1, \cdots, c\}$, $\mathbf{Set}_Y = \{Set_i : i \in \{1, \cdots, c\} \wedge i \neq x\}$:



(a) $E_{(1,x)}$



(b) E_2 (c) E_3

Figure 6: Specifications for the graph colouring case study. (a) The probability of a robot to assume a colour is reduced if a neighbour had already selected it; (b) specification of the colours' priorities; (c) waiting period for colour change.

4. SUPERVISOR SYNTHESIS

A monolithic supervisor, S, is obtained by the synchronous composition of all free behaviour models and specifications into a single generator [11]. We will illustrate this process using the models for the graph colouring strategy shown in Figures 5 and 6. First, all free behaviour models are composed into a single generator:

$$G = G_1 || G_2 || G_3. \tag{10}$$

All specifications are composed into a single generator:

$$E = E_{(1,1)} || \ldots || E_{(1,c)} || E_2 || E_3. \tag{11}$$

E and G are composed together into a target language:

$$K = G || E. \tag{12}$$

Finally, the monolithic supervisor S is the maximal controllable sub-language of K. S is a generator for which *bad* states and any states from which bad states can be reached through a sequence of uncontrollable events, are removed. A bad state is a state in the supervisor in which an uncontrollable event is denied from occurring (according to the specifications) but physically possible (according to the free behaviour models). The supervisor is non-admissible if it contains bad states. The state is referred to as bad, as the uncontrollable event cannot be disabled by the supervisor. The operation that realises the removal of bad states and also guarantees that all states can be reached (accessible) and can reach a marked state (co-accessible) is called *SupC*. Therefore, the monolithic supervisor is given by:

$$S = SupC(G, K). \tag{13}$$

A local modular supervisor explores the modularity of the free behaviour models and the specification to synthesise supervisors that are potentially smaller than the monolithic supervisor in the number of states and transitions [13]. This is done by creating a local modular supervisor for each specification based on a local free behaviour. The local free behaviour model of a specification is obtained by composing only the free behaviour models that contain the events used

[3]Note that during synchronous composition, the probabilities defined in different specifications will get multiplied and normalised.

in the specification. The local free behaviour model, G_x^{loc}, are:

$$\begin{array}{lcl} G_1^{loc} & = & G_1||G_2 \\ G_2^{loc} & = & G_1 \\ G_3^{loc} & = & G_1||G_3 \end{array} \quad (14)$$

Differently from the monolithic approach, specifications are not composed together. Instead target languages, K_x^{loc}, are obtained for each specification, as:

$$\begin{array}{lcl} K_{(1,i)}^{loc} & = & E_{(1,i)}||G_1^{loc} : \forall i \in \{1,\cdots,c\} \\ K_2^{loc} & = & E_2||G_2^{loc} \\ K_3^{loc} & = & E_3||G_3^{loc}. \end{array} \quad (15)$$

Similarly to the monolithic approach, bad states must be removed. Each target language results in a local modular supervisor, for the current case study they are:

$$\begin{array}{lcl} S_{(1,i)}^{loc} & = & SupC(G_1^{loc}, K_{(1,i)}^{loc}) : \forall i \in \{1,\cdots,c\} \\ S_2^{loc} & = & SupC(G_2^{loc}, K_2^{loc}) \\ S_3^{loc} & = & SupC(G_3^{loc}, K_3^{loc}). \end{array} \quad (16)$$

5. IMPLEMENTATION

The use of probabilistic deterministic finite generators under the pSCT framework requires two changes in the implementation, previously presented in [21]. First, the memory representation must include the probabilities of each controllable transition. Second, the choice between enabled controllable events must take into account the probability of each related transition in the current state.

5.1 Memory representation

Consider the probabilistic supervisor shown in Figure 7(a). Figures 7(b–c) illustrate the data structure that stores the supervisor in memory. In Figure 7(b) the representation of the partial transition function, δ, is shown [22]. Each state is represented by a block of this data structure. Each block describes all output transitions from that state. The first byte of each block 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. As uncontrollable events do not have an associated probability the probabilities of all controllable events are stored in a separate data structure (see Figure 7(c)). Each state is represented by a block of this data structure. Each block describes the probabilities of output transitions triggered by controllable events from that state. The first byte of each block is the amount of output of controllable transitions (o_c). It is followed by o_c sets of 2 bytes, where each set represents a transition's probability. The event of each set can be inferred from the partial transition function (δ) representation as each set is stored in the same order (see Figure 7b).

5.2 Probabilistic generator player

The probabilistic generator player (pGP) executes the generators realising the supervisors. We modify the traditional generator player presented in [21] to incorporate the calculation of the probabilities of multiple local modular supervisors. Probabilities of local modular supervisors are computed at run-time for the specific current state. The joint probability is calculated as defined in Equation 9 and it is normalised as defined in Equation 6. The monolithic supervisor can be normalised at the time it is being synthesised (prior to run-time). However, the normalisation for local modular supervisors can only occur after all synchronisation operations are performed, as, in general:

$$Norm(S) \quad = \quad Norm(S_1^{loc}||\cdots||S_i^{loc}) \quad \neq \quad Norm(S_1^{loc})||\cdots||Norm(S_i^{loc}). \quad (17)$$

For that reason, the probabilistic generator player needs to incorporate the calculation of the normalised probabilities.

6. EXPERIMENTS

Experiments are performed to validate the implementation of our pSCT model. In particular, they test whether the modelled specifications match with the synthesised control logic, as observed during the trials. Video recordings from all experimental trials and additional resources (models and the used source code) can be found in the electronic supplementary material [23].

The experiments took place on a two-dimensional glass-floored arena. We performed two sets of experiments. The first set is composed by trials using 25 Kilobot robots, distributed on a 5×5 grid. Twelve trials were performed, each lasting 25 minutes. To test the scalability of the approach we performed a second set of experiments using 100 Kilobot robots, distributed on a 10×10 grid. Two trials were performed, each lasting 45 minutes.

Robots are positioned on the grid in such way that their communication range only reaches other robots in the next or previous vertical and horizontal positions (so called Manhattan neighbourhood) but not in the diagonal. Therefore, robots in the middle of the grid have four neighbours, the four corner robots only have two neighbours, and the remaining robots in the border of the grid have three neighbours. The optimum solution for this case requires the use of two different colours and forms a checkered pattern. Robots are initially programmed with a code to assist the positioning process. The program changes the colour of the RGB led according to the number of neighbours a robot has.

We synthesised the supervisor using $c = 4$ (number of available colours), which resulted in a monolithic supervisor comprised of 240 states and 2480 transitions. By contrast, for the local modular approach, all supervisors collectively comprise only 20 states and 220 transitions. The use of the local modular approach corresponds to a reduction by 91.7 % and 91.1 % in states and transitions, respectively, compared to the monolithic approach.

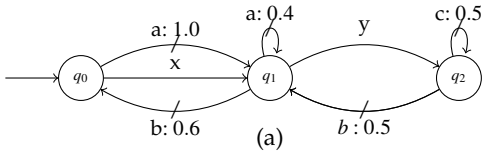
Figure 8 shows snapshots taken from one of the experimental trials with 25 robots[4].

Figure 9 shows snapshots taken from one of the experimental trials with 100 robots.

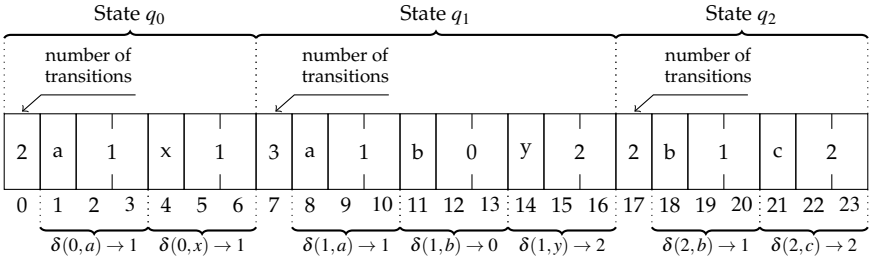
To evaluate the performance of the controller we measured the proportion of robots using up to 2 colours, $\varphi_{colours:2}$, which is known to be the optimal configuration for the experimental setup. We also measured the proportion of the connections among neighbours with different colours, $\varphi_{connections}$.

The proportion of robots using up to 2 colours for the 25 robots trials is shown in Figure 10 and the proportion of connections among neighbours with different colours is shown in Figure 11.

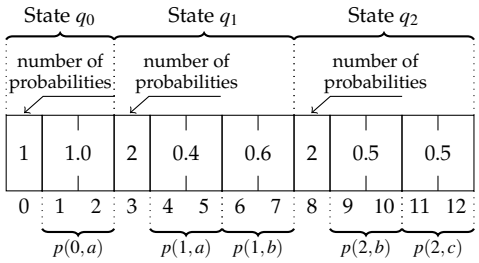
[4]Environment lights were kept off to facilitate the recording of the experiment.



(a)

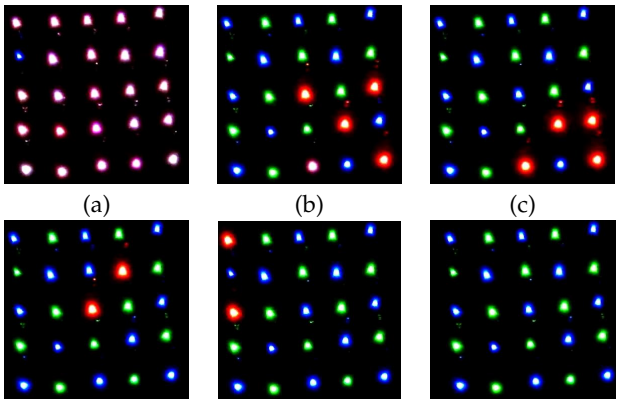


(b)



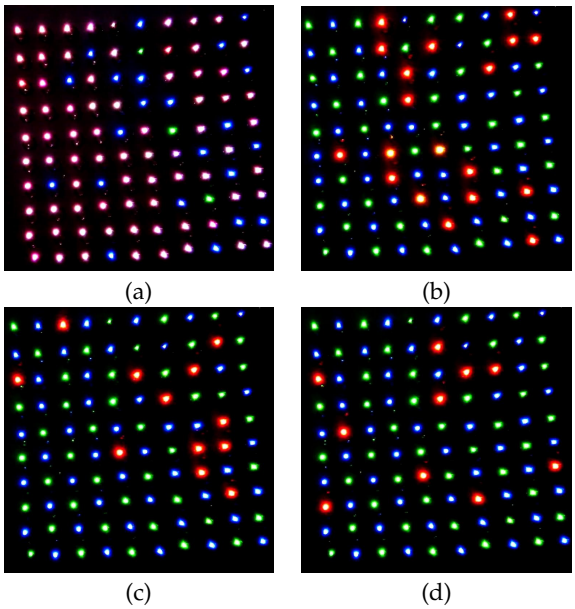
(c)

Figure 7: The memory representation of a probabilistic generator. (a) The probabilistic generator; (b) the partial transition function representation [22]. 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. (c) Representations of the probabilities of controllable transitions.



(a) (b) (c) (d) (e) (f)

Figure 8: A sequence of snapshots of one of the 12 trials where 25 Kilobots performed the distributed graph colouring algorithm: Photos (a-f) show the experiment after 0 s, 300 s, 600 s, 900 s, 1200 s, and 1500 s.



(a) (b) (c) (d)

Figure 9: A sequence of snapshots of one of the two trials where 100 Kilobots performed the distributed graph colouring algorithm: Photos (a-d) show the experiment after 0 s, 900 s, 1800 s, and 2700 s.

The proportion of robots using up to 2 colours for the 100 robots trials is shown in Figure 12 and the proportion of connections among neighbours with different colours is shown in Figure 13.

Ideally, the robot's communication range would allow it to form connections only with their immediate neighbour

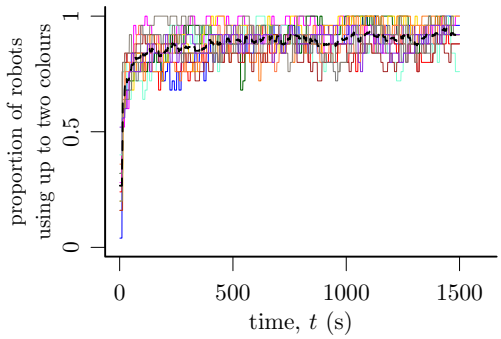


Figure 10: The proportion of robots using up to 2 colours in the 25 robots trials. Each coloured line represents one experimental trial. The thick black dashed line indicates the mean.

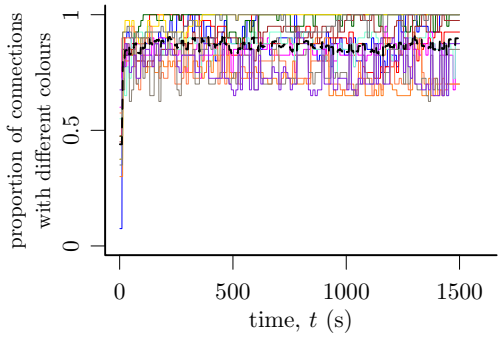


Figure 11: The proportion of the connections among neighbours with different colours in the 25 robots trials.

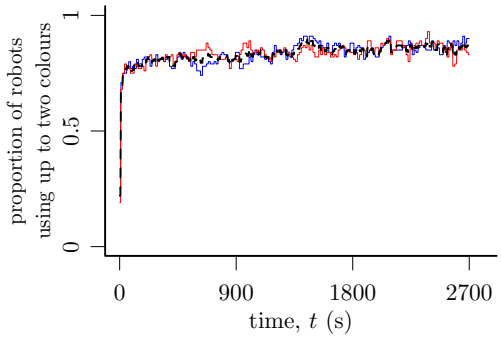


Figure 12: The proportion of robots using up to 2 colours in the 100 robots trials.

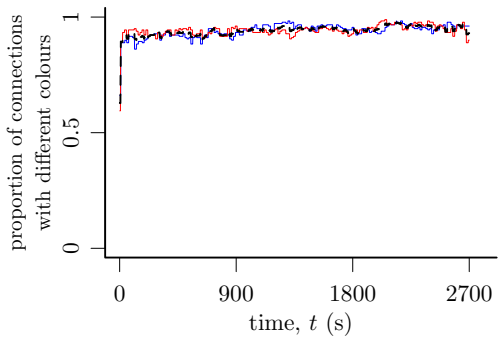


Figure 13: The proportion of the connections among neighbours with different colours in the 100 robots trials.

on the vertical or horizontal line. However, due to noise affecting the infrared communication among robots, additional connections were occasionally formed or broken. In the last minute of the 25 robots trials, we obtained an averaged success rate of 87% regarding $\varphi_{connections}$ and 93% regarding $\varphi_{colours:2}$. The first minute of execution, just after the initial setup, presented an averaged success rate of 68% regarding $\varphi_{connections}$ and 52% regarding $\varphi_{colours:2}$. For the 100 robots trials, we obtained an averaged success rate of 95% regarding $\varphi_{connections}$ and 87% regarding $\varphi_{colours:2}$ by the last minute of the trials. The success rate in the first minute of the 100 robots trials was 87% regarding $\varphi_{connections}$ and 67% regarding $\varphi_{colours:2}$. The results suggest that the strategy reaches solutions that optimise both measured metrics simultaneously. The strategy succeeds in rapidly converging towards these solutions, though a small percentage of errors remain.

7. CONCLUSION

In this paper, we proposed a probabilistic supervisory control theory (pSCT) framework. It uses a form of probabilistic generators to model probabilistic processes and, thereby, can prevent livelocks or indefinitely repetitive behaviour. Furthermore, through decomposition into local modular supervisors, the automatically generated controller code is smaller than that for a monolithic supervisor and is thus more likely to be applied successfully to a swarm robotic system.

To illustrate the advantages of the proposed framework, we presented a case study where the robots distributively and locally search for a solution to the graph colouring problem using a strategy modelled with pSCT. The local modular supervisors were collectively 91% smaller than the monolithic supervisor, both regarding the total number of states and state transitions. The generated code was deployed on physical swarms of 25 and 100 Kilobots, and systematic experiments were conducted. The results demonstrate that probabilistic solutions can be modelled successfully with pSCT. Future work could explore the use of pSCT in a variety of scenarios, for example, for controlling large groups of animals [24].

ACKNOWLEDGMENT

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. A.B. Leal wishes to thank CAPES and UFSC by post-doctoral fellowship (PNPD/CAPES - Graduate Program in Automation and Systems Engineering - PPGEAS/UFSC). This research was supported as well by the Engineering and Physical Sciences Research Council (Grant No. EP/J013714/1). The authors are thankful for the proofreading and feedback from Natalie E. Wood.

REFERENCES

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

[2] J. C. Barca and Y. A. Sekercioglu, "Swarm robotics reviewed," *Robotica*, vol. 31, no. 3, pp. 345–359, 2013.

[3] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Flocking in fixed and switching networks," *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 863–868, 2007.

[4] C. Dixon, A. F. Winfield, M. Fisher, and C. Zeng, "Towards temporal verification of swarm robotic systems," *Robotics and Autonomous Systems*, vol. 60, no. 11, pp. 1429 – 1441, 2012.

[5] M. Brambilla, A. Brutschy, M. Dorigo, and M. Birattari, "Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking," *ACM Transaction on Autonomous and Adaptive Systems*, vol. 9, no. 4, pp. 17:1–17:28, 2015.

[6] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari, "Automode-`Chocolate`: automatic design of control software for robot swarms," *Swarm Intelligence*, vol. 9, no. 2–3, pp. 125–152, 2015.

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

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

[9] A. Vieira, E. Santos, M. De Queiroz, A. Leal, A. DE Paula Neto, and J. Cury, "A method for plc implementation of supervisory control of discrete event systems," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 1, pp. 175–191, 2017.

[10] M. Queiroz and J. Cury, "Modular supervisory control of large scale discrete event systems," in *Proceedings of International Workshop on Discrete Event Systems (WODES)*. Berlin, Germany: Springer, 2000, pp. 103–110.

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

[12] M. Queiroz and J. Cury, "Modular control of composed systems," in *Proceedings of the 2000 American Control Conference*. Piscataway, NJ: IEEE, 2000, pp. 4051–4055.

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

[14] V. Pantelic, S. M. Postma, and M. Lawford, "Probabilistic supervisory control of probabilistic discrete event systems," *IEEE Transactions on Automatic Control*, vol. 54, no. 8, pp. 2013–2018, 2009.

[15] V. Pantelic, M. Lawford, and S. Postma, "A framework for supervisory control of probabilistic discrete event systems," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 477–484, 2014.

[16] C. Yoo, R. Fitch, and S. Sukkarieh, "Provably-correct stochastic motion planning with safety constraints," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe,Germany, May 6-10, 2013*, 2013, pp. 981–986.

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

[18] D. P. Dailey, "Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete," *Discrete Mathematics*, vol. 30, no. 3, pp. 289–293, 1980.

[19] L. P. Pinheiro, Y. K. Lopes, A. B. Leal, and R. S. U. Rosso, "Nadzoru: A software tool for supervisory control of discrete event systems," in *Proc. of the 5th International Workshop on Dependable Control of Discrete Systems (DCDS)*, vol. 5, 2015.

[20] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors." in *ICRA 2012*. Piscataway, NJ: IEEE, 2012, pp. 3293–3298.

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

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

[23] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Electronic Supplementary Material — Probabilistic Supervisory Control Theory (pSCT) Applied to Swarm Robotics," 2016. [Online]. Available: http://naturalrobotics.group.shef.ac.uk/supp/2016-008/

[24] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. M. LaValle, "Controlling wild bodies using linear temporal logic," in *Proceedings Robotics: Science and Systems*. Cambridge, MA: MIT Press, 2011, pp. 17–24.