
The University Of Sheffield.

# Supervisory Control Theory for Controlling Swarm Robotics Systems

**Yuri Kaszubowski Lopes**

*Supervisors:*
Dr Roderich Groß
Dr Tony J. Dodd

A Thesis Submitted for the Degree of
Doctor of Philosophy

9th December 2016

To obtain, something of equal value must be lost. That is Alchemy's first law of Equivalent Exchange.

*Hiromu Arakawa*
*Fullmetal Alchemist, Vol. 1*

# Abstract

Swarm robotics systems have the potential to tackle many interesting problems. Their control software is mostly created by ad-hoc development. This makes it hard to deploy swarm robotics systems in real-world scenarios as it is difficult to analyse, maintain, or extend these 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.

This thesis studies the application of the supervisory control theory (SCT) framework in swarm robotics systems. SCT is widely applied and well established in the manufacturing context. It requires the system and the desired behaviours (specifications) to be defined as formal languages. In this thesis, regular languages are used. Regular languages, in the form of deterministic finite state automata, have already been widely applied for controlling swarm robotics systems, enabling a smooth transition from the ad-hoc development currently in practice. This thesis shows that the control code for swarm robotics systems can be automatically generated from formal specifications.

Several case studies are presented that serve as guidance for those who want to learn how to specify swarm behaviours using SCT formally. The thesis provides the tools for the implementation of controllers using formal specifications. Controllers are validated on swarms of up to $600$ physical robots through a series of systematic experiments. It is also shown that the same controllers can be automatically ported onto different robotics platforms, as long as they offer the required capabilities. The thesis extends and incorporates techniques to the supervisory control theory framework; specifically, the concepts of global events and the use of probabilistic generators. It can be seen as a step towards making formal methods a standard practice in swarm robotics.

# Acknowledgments

Não há outra maneira de se iniciar meus agradecimentos senão em minha língua mãe, pois muitos dos que contribuirão para tornar possível este trabalho escrito em língua estrangeira talvez não possam lê-lo, mas nada existiria sem eles. Primeiramente eu gostaria de agradecer as duas pessoas mais importantes da minha vida, os meus pais, Almir Lopes e Denise Kaszubowski Lopes, por toda a compreensão, amor, valores, ensinamentos e carinho durante toda minha vida. E também a toda minha família, em especial aos meus tios Marlon Manoel de Souza e Dirlei Kaszubowski de Souza, meu primo e grande amigo Eric Kaszubowski de Souza, meus queridos avós, Maria de Lourdes Kaszubowski, Thadeu Kaszubowski (In Memoriam) e Maria de Lordes Pereira Lopes. Meu agradecimento ao povo brasileiro, que por intermédio da CAPES financiaram meu doutorado e ao povo catarinense que através de seus impostos financiaram minha graduação e mestrado na universidade do estado de Santa Catarina.

I would like to express my gratitude to a number of people who provided expert advice, support, and guidance during my PhD. My supervisors, Dr. Roderich Groß and Professor Tony J. Dodd who spared no efforts to assist my research. My scholarship tutor in Brazil, Dr. André Bittercourt Leal. My examiners, Professor Visakan Kadirkamanathan and Pedro Lima.

I also would like to acknowledge the support of several members of staff from the University of Sheffield. Especially the staff from the Automatic Control and System Engineering department. The staff members of Sheffield Robotics, in particular to Ana MacIntosh, Louise Caffrey, and Michael Port.

I am thankful to my lab mates at the Natural Robotics Lab, for all of the friendship, advice, support, and the friendly environment for the discussion of ideas: Melvin Gauci, Jianing Chen, Wei Li, Christopher Parrott, Fernando Perez-Diaz, Gabriel Kapellmann Zafra, Matthew Doyle, Nicole Salomons, João Vasco Marques, Anıl Özdemir, Isaac Vandermeulen, Yue Gu, and in particular to Stefan Trenkwalder for his collaboration with my work. I am thankful to my other PhD student fellows from Sheffield Robotics: Haoyu Zhang, James Douthwaite, Christina Georgiou and especially to Natalie Elizabeth Wood.

Adapting to live in another country could have been extremely challenging, so I am thankful that I was able to find a piece of Latin America with this amazing Mexican

group and their friends, who made the stay in Sheffield so enjoyable. Moitısimas gracias, José Solis Cordova, Daniela Miranda, Francisco Ochoa Cardenas, Nora Elvia Manzo Flores, Ariel Cano, Alejandro Vidal Rosas, Yessica Garcia de Vidal, Vicktor Cedeno Campos, Carlos Luna, Erick Noe Amezquita Lucio, Yang Zhang, and Matei Neagu.

To other friends I made in Sheffield during this journey, especially to Victor Borges, Mike Gransbury, Rebecca Gransbury, and Bernadette Hill.

Finally, a thank you to the people back home who supported me on this journey. My dearest professors from the Santa Catarina State University, without their teachings I would never be prepared for this PhD. Especially to Roberto Silvio Ubertino Rosso Junior, André Bittencourt Leal, Carlos Norberto Vetorazzi Junior, Claudio Cesar de Sá, Salvador Antonio dos Santos, Alexandre Gonçalves Silva, Ricardo Ferreira Martins, Omir Correia Alves Junior, Marcelo da Silva Hounsell, Ademir Nied, and Pedro Bertemes Filho. To my friends and colleagues from the Santa Catarina State University Lucas Hermann Negri, Romulo Amorim Bahiense, Gabriel Hermann Negri, Guilherme Jarentchuk, André Diego Piske, Guilherme Espindola, Thiago Oliveira, Renan Sebem, Eduardo Harbs, Rodrigo Trentini, and Yujuan Wang.

Without you I would never have had the opportunities to find my way to this milestone.

To planet Earth, for its generous hospitality.

*Yuri Kaszubowski Lopes*

# Nomenclature

| | |
|---|---|
| CCW | Counter-clockwise |
| CW | Clockwise |
| DES | Discrete event system |
| DFA | Deterministic finite state automata |
| FSM | Finite state machines |
| HST | Hybrid systems theory |
| OHP | Overhead programmer |
| PDFA | Probabilistic deterministic finite state automata |
| PFSM | Probabilistic finite state machine |
| PFSM | Probabilistic finite state machines |
| pGP | Probabilistic generator player |
| PLC | Programmable logic controller |
| pSCT | Probabilistic supervisory control theory |
| SCT | Supervisory control theory |
| UML | Unified modelling language |
| VPB | Virtual physics-based |

# List of Figures

# List of Tables

# Contents

# 1

# Introduction

To improve the world around us, a substantial amount of commodities are being produced every day, and this has driven humans to change the way labour is tackled. Hand tools have been a major advance, reducing much of the human effort in performing tasks. Further mechanisation, such as the use of powered tools and advanced machinery, have reduced, even more, the physical effort required for the manufacture of goods. Machines and robots are now able to perform a range of tasks, releasing humans from those that are repetitive, dangerous, and tedious.

Robots are not only able to assist humans in several tasks but they can even outperform them in quality and speed. Robots can provide companionship, extend human abilities, they can be built to be many times stronger than humans, and can do tasks more quickly, or make decisions beyond human capabilities. Multiple robots can be coordinated to work collaboratively, and more recently, to self-organise as a collective. In the future robots will assist humans to tackle labour by the spontaneous self-organised collaboration between groups of robots and humans. The question on how this can be achieved has aroused the interest of many researchers.

A promising answer comes from nature. Since the dawn of time, humans have observed nature for inspiration in developing new technologies. Recently, the collective behaviours found in nature have fomented a new horizon on how we can build robotic systems that spontaneously self-organise and collaborate to perform tasks. Some of the most prominent methods that arose from observing the social interactions of animals are grouped under the term swarm intelligence. Swarm intelligence techniques include well known meta-heuristics such as stochastic diffusion search [2], ant colony optimisation [3, 4], and particle swarm optimisation [5]. A new and prominent field of study, called *swarm robotics* [6, 7, 8, 9, 10, 11], was opened by translating swarm intelligence techniques, from a virtual population of agents that search for a solution for an opti-

misation problem, to real-world scenarios where agents are real robots cooperating to solve physical tasks.

Swarm robotics systems may accomplish tasks despite failures in some of the robots, and are typically designed so that their performance scales well with the number of robots. These properties can be useful in several applications [11]. However, much of the source code used to control swarm robotics systems is developed in an ad-hoc manner, meaning that the correctness of controllers is not easily verifiable [12] hindering its transition from the academic environment to real applications. Formal methods are an ideal tool for addressing these problems.

Formal methods is an umbrella term for a set of mathematical techniques, languages, and tools for the modelling, specification, and verification of systems [13]. A promising field of study within formal methods is formal languages.

Formal languages study the mathematical representation of languages. The theory of formal languages is concerned with the classification, patterns, features, properties, relations, specifications, and recognition of languages [14]. The field started paying attention to such characteristics in natural language, mainly on the syntactic aspects, but later this was extended and applied to other areas [14]. For example, compilers make use of formal languages to specify how a computer program can be written. Formal languages are applied in the recognition of patterns and in the definition of communication standards. Formal languages are a useful tool to model systems; allowing a formal mathematical representation of systems that can be understood by computers and humans.

The supervisory control theory (SCT) [15, 16, 17]—or Ramadge-Wonham framework—is a framework for the synthesis of controllers in the form of supervisors. Many works in the manufacturing context have contributed to render the SCT as a reliable practice [18, 19, 20, 21, 22, 23, 24, 25, 26, 27]. 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. Regular languages, recognised by deterministic finite state automata (DFA) (see Section 2.2.1 and generated by generators (see Section 2.3.1), are the most common approach used in the literature to express formal languages to be applied with the SCT framework.

## 1.1 Motivation

The motivation for this work emerges from the potential uses of swarm robotics, and the question about how formal methods can assist in the transition of swarm robotics systems from the academic environment to real-world applications. Swarm robotics explores collaboration among multiple individuals, an approach consistently found in nature. Swarms of insects, colonies of ants, flocks of birds, and several other animals use collaboration to perform their day to day activities. We humans are not different and often collaborate to perform labour. Swarm robotics has the potential to formulate intelligent solutions for several problems by combining robotics with the social aspects of collaboration.

Future applications of swarm robotics systems include: medical [28], patter formation [29, 30], patrol-inspection [31, 32], transportation [33, 34], among others. In these applications, a single specialised robot may not be used as some of the requirements can exclusively be met by swarms of robots. For example, in a search and rescue environment, a swarm of small robots could be used to search and access obstructed locations and when a victim is found the robots could self-assemble to combine strength to transport the victim to a safer location [35].

Formal methods have the potential to increase the reliability of the behaviours exhibited in swarm robotics systems. Once the desired behaviour of a swarm is formally expressed the specification can be subject to verification and tests, for example, model checking techniques. The formal specification can also serve as a documentation of the behaviour.

In [36] interviews with nuclear engineers exposed for a short time to the use of formal methods (in the form of "Statecharts") resulted in the following two comments:

> " They felt that they could eventually come to an agreement that the Statechart specification correctly described the system and did not feel that they would have the same confidence with an English document. "

And:

> " Once the nuclear engineers had experience with one or more of the formal specification notations, they said they would never trust a natural language specification again. "

This work focuses on the use of regular languages—that can be realised by generators (a type of finite state machines)—within the SCT framework. The use of regular languages may enable a smooth transition between the ad-hoc development using finite state machines—commonly employed by the swarm robotics community—and the use of formal languages that can automatically generate the control code for swarm robotics systems. The motivation of this work is the potential acceptance of SCT using finite state machines by the swarm robotics community that can drive the use of formal methods as a standard practice in the field.

## 1.2 Problem definition

Designing the control logic for a swarm of robots is a challenging problem. Each robot in the swarm typically executes an identical program that 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 hard to maintain, analyse, or verify.

Formal methods help in addressing these problems (for example, see Knight et al. 36). However, even when formal methods are used, it is not guaranteed that the final source code would accurately represent the specifications. This uncertainty is due to the fact that the source code has been obtained in a manual process as automatic code generation is not a standard practice in swarm robotics.

Furthermore, a cultural resistance to applying formal methods can be observed [36]: they may be perceived as an additional step that does not add value to the final product, prolongs its development cycle, introduces undesired complexity, and their integration is often impeded by the lack of appropriate tools.

We propose the application of SCT to the domain of swarm robotics. We investigate how to formally model the capabilities of and specifications for swarm robotics systems; how to automatically generate the controllers (including source code) for the in-

dividual robots in the swarm; and how SCT can be extended to provide useful methods for swarm robotics systems.

## 1.3 Aim and objectives

The aim of this thesis is to investigate the use of regular languages for the synthesis and control of swarm robotics systems within the supervisory control theory framework.

The specific objectives of this work are:

- To show, through several case studies (either novel or found in literature), how swarm robotics solutions can be modelled using the SCT and regular languages expressed by generators.
- Investigate how the SCT can reduce the amount of ad-hoc code.
- Investigate how the SCT can cope with different swarm robotics platforms.
- Investigate the SCT's support for extensions.
- Examine the performance of the approach using swarms of physical robots.

## 1.4 Preview of contributions

The contributions of the thesis are:

- To use supervisory control theory for formally developing controllers for swarms of autonomous robots and illustrate the modelling process using regular languages (generators) as the formal language in a range of case studies;
- To adapt the full supervisory control theory framework [15, 22, 1], from the modelling to the software implementation, and validate its use with swarms of up to 600 physical robots;

- To adapt an open source software tool to automatically generate the control software for two established swarm robotics platforms (the e-puck [37] and the Kilobot [38]), and to demonstrate that the same synthesised supervisor can be directly executed on swarms of different platforms as long as they offer the required capabilities;

- To compare three existing supervisory control synthesis methods: monolithic [15], modular [16], and local modular [20, 19, 22] in several swarm robotics' case studies;

- To extend the SCT framework by introducing the concept of global events and validate the extension in four case studies with swarms of physical robots;

- To propose a probabilistic SCT (pSCT) framework, which combines the concept of probabilistic generators [39, 40] with support for marked states[1] and the synthesis of local modular supervisors [20, 19, 22], and to validate pSCT in one case study with swarms of physical robots.

## 1.5 Publications

This thesis provides original contributions to scientific knowledge established by the author's own work. Five papers for academic journals and conferences have been produced based on the work presented in this thesis:

1. LOPES, Y. K., TRENKWALDER, S., LEAL, A. B., DODD, T. J., GROSS, R. Supervisory control theory applied to swarm robotics. Swarm Intelligence, 10(1):65–97, 2016.

2. LOPES, Y. K., LEAL, A. B., DODD, T. J., GROSS, R. Application of Supervisory Control Theory to Swarms of e-puck and Kilobot Robots. In: M. Dorigo, et al. (Eds.), International Conference on Swarm Intelligence - ANTS, 2014, volume 8667 of LNCS, pages 62-73. Berlin: Springer. Finalist of Best Paper Award.

3. PINHEIRO, L.P., LOPES, Y.K., LEAL, A.B., ROSSO Jr, R.S.U. Nadzoru: A software tool for supervisory control of discrete event systems. In: Proceedings of the

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

5th International Workshop on Dependable Control of Discrete Systems - DCDS, volume 5, 2015, pages 182-187.

4. LOPES, Y. K., TRENKWALDER, S., LEAL, A. B., DODD, T. J., GROSS, R. Probabilistic Supervisory Control Theory (pSCT) Applied to Swarm Robotics (to appear).

5. LOPES, Y. K., TRENKWALDER, S., LEAL, A. B., DODD, T. J., GROSS, R. Global Discrete Events in the Supervisory Control of Swarm Robotics (in preparation).

The author presented Publication 2 as a full paper at the International Conference on Swarm Intelligence - ANTS 2014, held in Brussels, Belgium.

The contents of Chapters 3 and 4 of this thesis have been previously published in Publications 1 and 2. The materials in Publications 5 and 4 are based on the contents of Chapters 5 and 6 of this thesis, respectively. Part of the contents related to the software tool nadzoru of Publication 3 represents additional material added to Chapter 4 of this thesis. The introductory content, background, and related work from all of these five publications have contributed to Chapters 1 and 2 of this thesis.

During the course of the author's PhD project, another publication, that is not featured in this thesis, has been done with his contribution:

1. TRENKWALDER, S., LOPES, Y.K., KOLLING A., CHRISTENSEN, A.L., PRODAN, R., GROSS, R. OpenSwarm: An Event-Driven Embedded Operating System for Miniature Robots. In: International Conference on Intelligent Robots and Systems - IROS 2016.

## 1.6 Thesis outline

This thesis' structure takes the form of seven chapters, including this introductory chapter. The remaining part of this thesis proceeds as follows:

- Chapter 2 presents the related work and reviews the background theory. It starts with an overview of the swarm robotics fields in Section 2.1. Specifically, methods for the design, modelling, analysis, and control of swarm robotics systems

are presented in Section 2.1.1. In Section 2.1.2 several works that consider the use of formal methods in swarm robotics are presented. In Section 2.1.3 two physical swarm robotics platforms are presented: the Kilobot [38], and the e-puck [37]. The theory of formal languages and automata is introduced in Section 2.2. In Section 2.3 the supervisory control theory (SCT) for the control of discrete event systems (DES) is presented. The concepts of generators (Section 2.3.1), free behaviour models (Section 2.3.2), and control specifications (Section 2.3.3) are defined. Section 2.3.4 presents the steps for the synthesis of supervisors using the SCT framework, Section 2.3.5 discuss the control implementation, and Section 2.3.6 overviews the application of SCT. Finally, probabilistic generators are overviewed in Section 2.3.7.

- Chapter 3 presents how to model the capabilities, how to specify the desired behaviour, and how to synthesise controllers in the form of supervisors for swarm robotics systems using the SCT framework. It introduces five case studies and the model of the formal specification of the desired behaviour in Section 3.1. It starts with a didactic case, the orbit case study, in Section 3.1.1 and the segregation case study, in Section 3.1.2. Further on, in the next Chapter, these two cases studies are implemented in different swarm robotics platforms with source code generated from the same formal specification. In Sections 3.1.3 and 3.1.4 two cases from literature are formally specified: the aggregation and the clustering of objects. The scalability of the approach is checked by the group formation case study presented in Section 3.1.5. Finally, in Section 3.2 the formal controllers (supervisors) are synthesised using different methods and the results are compared in Section 3.2.4.

- Chapter 4 discuss the implementation of formal controllers from the synthesised controllers obtained in the previous chapter and presents the results obtained from systematic experimental trials performed on physical robots. The automatic code generation, which is a compact representation of the synthesised supervisor, and the required implementation are presented in Section 4.1. The experiments are described and results are reported in Section 4.2.

- Chapter 5 presents the concept of global events that abstracts communication among robots in the swarm. In Section 5.1 the concept of global events for the supervisory control of swarm robotics is introduced. Four case studies are presented in Section 5.2, illustrating the concept of global events. It is shown how

global events can be used to make the swarm cooperatively respond to situations detected or observed by other robots (in Sections 5.2.1 and 5.2.2) and how robots can make a joint decision to avoid conflicts (in Section 5.2.3) or to represent a consensus (in Section 5.2.4, where robots perform synchronous movement). We detail the automatic code generation for the controllers that support global events in Section 5.3 and we describe the experimental methods and results in Section 5.4.

- Chapter 6 presents how we incorporate probabilistic generators to formulate probabilistic controllers in a probabilistic supervisory control theory (pSCT) framework. In Section 6.1 it is shown how probabilistic generators within the pSCT framework can solve the choice problem (see [18]). From the adaptation of the definitions of probabilistic generators found in literature, shown in Section 6.2, we define a set of operations for the synthesis of probabilistic supervisors that can cope with multiple models for the definition of the robots' capabilities and specifications, in Section 6.2.1. We present a study case based on the graph colouring problem in Section 6.3. We synthesise a probabilistic controller using pSCT and in Section 6.4 we detail its implementation on physical robots using automatic code generation. Our controller, automatically obtained, can distributively compute a solution for this problem, as shown in Section 6.5 through experimental trials.

- Chapter 7 concludes the thesis by summarising research findings, their implications, and highlighting the significance of the findings. The chapter finally provides recommendations for further research work.

# 2

# Background and related work

This chapter first reviews the literature on swarm robotics including related work on the application of formal methods. It then reviews formal languages and Supervisory Control Theory (SCT).

## 2.1 Swarm robotics

Swarm robotics studies systems composed of a large number of autonomous robots that interact with each other and cooperate to achieve certain goals. Swarm robotics systems are characterised by restricted communication among the individuals within the swarm, use of local information, decentralised control, and the emergence of global behaviour [7]. Robotic swarms tend to be robust, flexible, and scalable. Such properties may prove useful in many real-world applications [11].

The term swarm intelligence was originally introduced by [8] to describe cellular robotic systems. Later the term has been adopted in the context of social insects and optimisation [9]. [41] defines swarm intelligence as a collective behaviour that emerges when many simple individuals combine their separate behaviours. Swarm robotics can be seen as the application of swarm intelligence concepts to collective robotics [42].

Some authors argue that the term swarm intelligence is too advanced to be used to describe modest (in terms of what robots can achieve) groups of robots [8]. A definition of swarm robotics is given by [9]:

> "Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment."

According to [11] the main characteristics of the robots from a swarm robotics system are: (1) they are autonomous, (2) they are located in the environment and can act to modify it, (3) they have local sensing and communication capabilities, (4) they do not have access to global knowledge and to centralised control, and (5) they cooperate to accomplish a given task.

In recent years swarm robotics has been extensively studied as discussed in reviews of this field [43, 44, 42, 45]. Several research domains have been considered; for example, modelling, behaviour design, biologically inspired methods, communication, mapping and localisation, control approaches, reconfiguration of robotic systems, manipulation and transportation of objects, task allocation, motion coordination, and analytical studies [44, 42].

This thesis is mainly concerned with the control of swarm robotics systems, specifically, how controllers can be automatically obtained from formal specifications, how we can verify safety conditions of such specifications, and how the controller can be implemented on real robots.

### 2.1.1 Design, analysing, and control

According to [11], the design of swarm robotics systems can be categorised into automatic design or behaviour-based design. Automatic design concerns the application of methods that automatically generate behaviours. Examples of automatic design methods are reinforcement learning [46] and evolutionary robotics [47]. Behaviour-based design concerns the development of system behaviour, primally by developers [11]. Behaviour-based design can make use of several techniques such as finite state machine (FSM) [48] and virtual physics-based (VPB) designs such as potential fields [11].

The above binary classification of design methods seems however not to be an accurate representation of reality, as the classes of design methods may overlap. Automatic design methods usually require developer input, for example, in the form of a fitness function that expresses the desired behaviour [49], whereas techniques listed as behaviour-based design can be automatically obtained [50].

FSM are mathematical models where the state space of the system is a finite discrete set. FSM is a common choice to represent behaviour-based design in swarm robotics [38, 12]. They are usually implemented by switch-case statements. Transitions between

states occur abruptly and are usually related to a change in stimulus, and each state is usually related to a particular control routine.

In [12] a tool called RoboChart is presented, which automatically generates control code from a notation based on FSM and inspired by the unified modelling language (UML) [51]. RoboChart allows a specification to be described graphically or textually. The automatic code generation tool then provides an implementation based on this specification. Two case studies are presented: self-organised aggregation [52, 53] (see Section 2.1.3.2) and swarm taxis [54]. The method used by the authors presents similarities with the work presented in Chapters 3 and 4 of this thesis and previously published in [55] and [56]. However, RoboChart is shown to support only a single specification model, whereas SCT, used in this thesis, allows the decomposition of the specification model into several sub-systems, which is usually simpler to describe than a larger single specification. Another difference is that the control output is related to states and the input to transitions in RoboChart, whereas in this thesis, both, control input and control output, are related to transitions (the RoboChart's approach to controlling output could be implemented with self-loop transitions). Furthermore, the approach used in this thesis makes possible to distinguish the system state once a control output is triggered.

Probabilistic FSMs (PFSMs) are an extension of finite state machines (FSMs). In PFSM transitions are associated with probabilities, which can be constant or change over time [44, 11]. The application of PFSM in swarm robotics includes the work of [57] where the collective behaviour of aggregation was studied. In the aggregation task, all robots in a homogeneous environment have to group themselves [11]. [58] explores chain formation behaviours, in which the robots have to connect two points of interest in the environment. In another application, [59] applies PFSM to task allocation. Task allocation refers to the collective behaviour in which different tasks are distributed across the robots [11].

VPB design has its origin on how physics studies particles [11]. Robots are seen as virtual particles that interact with each other by exercising and being subject to virtual forces. Examples of VPB design in swarm robotics include the works of [60] and [61]. In these works, artificial potential fields are applied to represent the virtual forces from the environment. Obstacles are related to repulsive forces, and targets are related to attractive forces. Furthermore, the work of [61] goes beyond by associating robots to virtual forces, where these forces may represent social interactions.

[44] classifies the analysis of swarm robotics into four different approaches, which is

based on the modelling characteristics of the system. The first is *sensor-based*[1] modelling that is concerned with simulations of the system. The main components of the system modelled by the sensor-based approach are the sensors, actuators, and environment objects. The robot-to-robot and the robot-to-environment interactions are modelled only after these principal components were modelled.

*Microscopic* models are the second category. They consider robot-to-environment and robot-to-robot interactions separately. Behaviours are associated with states, and the transitions between states are related to internal events inside the robot and external events in the environment [44].

*Macroscopic* models are the third category. They consider the swarm as a whole and thus directly model properties of the entire system [11]. The system's macroscopic state describes the number of robots in a particular microscopic state. The change in the system's macroscopic state can be described by differential equations [44], representing the evolution over time of the ratio between the number of robots in a particular state and the total number of robots [11]. These equations capture the system's mean behaviour, however, because of finite-size effects, they may not accurately model small swarms. Master equations [62] can be applied used to capture the dynamics of the system's state distribution. They may, however, not be feasible for large swarms.

[11] highlights the work of [63], which models a clustering task where robots collect objects, as one of the first to apply differential equations. Another task where robots must cooperate to remove sticks out of holes also is analysed by using differential equations [64, 65]. A foraging task that considers the effect of interference is modelled using differential equations by [66]. In this case, authors apply the robot group size as a decreasing function to model the individual performance of the robots. Chain and aggregation formation is also explored using differential equations [67], while applying PFSMs.

In [62], authors studied for the first time the problem of predicting the yield of self-assembly systems. Given an initial state or input the yield is the final quantity of a desired outcome over an indefinite period. In the work of [62], the starting configuration is a group of $x_1$ mechanical equilateral triangle parts, $X_1$, which can magnetically connect with each other. One side of the triangle has a north magnet, another a south magnet, and the third has no magnet and therefore cannot establish any connection. Two single parts of type $X_1$ can connect and form another shape compound of two

[1] The term sensor-actuator-based modelling might be more accurate.

parts, $X_2$; another single part, $X_1$, can connect with $X_2$ and form a three parts component, $X_3$; and so on. The outcome is the predicted number, $x_6$, of 6 parts components, $X_6$, which forms a hexagon. The possible combinations can be expressed by chemical relations, for example:

$$2X \rightarrow X_2, X + X_2 \rightarrow X_3, X + X_3 \rightarrow X_4, 2X_2 \rightarrow X_4. \tag{2.1}$$

Authors applied difference equations to determine the mean value of $x_i : i \in \{1, \cdots, 6\}$ and the master equation to determine the probability distribution.

The fourth category of swarm robotics analysis approaches presented by [44], *cellular automata* (CA), can be considered a special case of microscopic modelling. The CA is composed of a single or multi dimensional grid of cells. A finite number of possible states is related to each cell in the grid. Cells can only interact with the neighbouring cells, these interactions occur locally and are regulated by a set of rules.

Whereas the distinction between microscopic and macroscopic seems clear, sensor-based and CA appear as subcategories of microscopic and macroscopic categories. A CA can be seen as a particular method of the macroscopic analysis, rather than a different category. Similarly, sensor-based approaches does not appear as a different approach, but as a specification of the microscopic modelling.

Even though swarm robotics emphasises decentralised approaches, the control of swarm robotics systems can also include centralised control. In the centralised control there is a single control agent [68, 45], which can be a robotic agent (leader) [69]. In the decentralised control agents operate locally [68, 45] and are completely autonomous [69]. In the decentralised control of swarm robotics systems, the robots are self-organising [42, 69].

Note that authors may either refer to decentralised control as distributed control (e.g., [69]) or have distinct meanings for both terms (e.g., [68]). In [68], the decentralised control is split into two categories: distributed architectures, in which all agents perform the same control strategy, and hierarchical architectures, in which the agents can be considered as locally centralised. On the other hand, in [69], the centralised control is split into two categories: strong centralisation, in which the central agent remains the same, and weak centralisation, in which multiple robots are allowed to become a leader. The study and experiments of these categories of control and the proper balance between centralised and distributed control are investigated by [70].

The design aspect is concerned with the methods used to produce swarm robotics system's behaviours. These behaviours can be expressed by models using a particular representation. Representations that allow the analysis of the models can be referred as formal methods. The analysis of the models can be performed, for example, at microscopic or macroscopic levels. Furthermore, the analysis can check properties of the system, for example, safety conditions. The control of swarm robotics systems tends to be decentralised, even though centralised implementations exist in the literature. In this thesis, we focus on the application of models represented by a specific formal method to automatically generate the control that adheres to the specifications. The next section reviews works that approach formal methods in swarm robotics.

### 2.1.2 Formal methods

Formal methods are languages, techniques, and tools which are based on mathematical principles for specifying and verifying hardware and software systems [13]. The formal specification of a system is defined by [13] as a process to describe the system behaviour and its desired properties. In this work, we split this definition into system modelling, which describes the system's possible behaviours (capabilities) and control specifications, which define the desired behaviour/properties. The specifications can than be subject to formal verification and analysis, which check whether a system holds certain properties [13]. This section overviews previous work that investigates the application of formal methods in swarm robotics.

According to [71] the use of formal methods can be classified by the level of rigour:

- Level 0: Absence of formal methods;
- Level 1: Use of mathematical notations and concepts: formal specifications are used to communicate, using a precise and compact notation. This helps to reduce ambiguities.
- Level 2: Application of formalised specification languages with tool support: formal languages are used to model the system in a more structured way. This makes use of some automated tools to perform checking and to simulate the system. Specifications can be used to produce a simulation or to prototype an implementation, automatically or with human assistance. Formal verification can prove specification properties and infer system's behaviour;

- Level 3: Full use of formal specification, proof checking, and automated theorem proving: Whereas proofs in level 2 are performed informally, level 3 provides automated theorem proving. Automated theorem proving can be obtained by using computer programs even though the processing cost can be high.

As previously presented, swarm robotics systems can be modelled at two levels of abstraction: microscopic and macroscopic. In [65], the authors present a unified framework focused on non-automatic design methods, which uses PFSM, for modelling a swarm robotics system at both microscopic and macroscopic levels. The authors present a case study where a swarm of robots cooperatively pull sticks out of the ground. This work contributes to the area by combining microscopic and macroscopic modelling under a more rigorous, unified framework, presenting a swarm robotics case using probabilistic modelling methods and describing abstraction steps.

As [72] argue, modelling a system at multiple levels (microscopic and macroscopic) can lead to inconsistencies between such levels. 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 an automated system analysis technique called model checking.

In the work of [73], 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.

In [74] Lyapunov analysis, singularity theory, and monotone dynamical systems were applied to study collective decision-making behaviour in groups of animals. The agent mechanisms were modelled at the microscopic level to characterise the steady-state behaviour in the honeybee nest site selection problem.

[75, 76] propose a method called property-driven design. It uses a discrete-time macroscopic-prescriptive model and model checking. 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. Finally, the desired robot swarm is implemented and tested in simulations and then with robots. In the

property-driven design, the swarm is described using properties and the robot has a prescriptive model. This is used to simulate and to be implemented on robots, and it is applied to perform a top-down design for robotics swarms using two case studies. However, the formal specification is used only to guide the development, such as a blueprint is used in the development of a building. The authors highlight that the use of such methods improves upon the traditional ad-hoc approach, which depends on the developer's ingenuity and expertise. However, the obtained model is used only as a guide for the implementations of both the simulated and the physical swarms. The 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 [50], 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 [50] and by human experts [77]. 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 [78]. However, a well established community-wide practice that is shared and empirical still not available [79].

In [80], methods for the evaluation of the expected value of a path integral for general Markov chains on finite state spaces are reviewed. Authors demonstrate the method application in the analysis of swarm robotics systems. The work concludes that interactions between the robots can be so complex that sometimes the only practical way of studying those systems is to use numerical simulations.

Petri-nets (PN) [81] are another formal approach to modelling the control software of swarm robotics systems. Petri-nets can represent formal languages and therefore are a suitable candidate to be used with the SCT-framework. In [82], the authors use PNs to coordinate the actions of a group of robots. A mechanism for verifying group plans prior to execution, and an online approach to detect and solve conflicts using PN is introduced. The PN based controller is, however, executed from a central computer, which communicates with the robots. In [83], PN models were used in multi-robotics

for task planning, plan execution, and plan analysis. It is shown how to detect properties such as boundedness, livelocks, and deadlocks in PNs. The authors extend a PN-based framework by introducing communication models and communication actions to model and analyse the execution of multi-robot tasks.

[84] presented the design and implementation of a mission control system for an autonomous under-water vehicle based on Petri-nets. The Petri-nets are used to specify and execute the desired autonomous vehicle mission. However, the Petri-nets are not used to model the solution, instead the user synthesises the solution using an imperative language and the result are Petri-nets.

As in the case for supervisors[2] represented by finite state automata, checking for the controllability (see section 2.3.4.2) of deterministic PN is decidable [85, 86]. The controllability was believed to depend on the uncontrollable marking notion defined in [85]. The problem was thus reduced to checking the existence of reachable uncontrollable markings. However, as shown in [86], such a definition is not correct, as there are controllable deterministic PNs with reachable uncontrollable markings based on the definition presented in [85]. An adjusted definition of uncontrollable marking solving this issue is presented by [86].

Temporal logic [87] has been applied to the modelling and analysing of swarm robotics systems [88]. It can be combined with model checking for formal verification [89, 90, 91]. 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 developed.

In [90] and [91], formal verification techniques are applied for analysing the emergent behaviours of swarm robotics. The authors used model checking, and temporal logic to verify whether all possible behaviours within the swarm will hold the desired properties. The method works by checking if temporal properties are satisfied. The paper concludes that the formal method of temporal logic can be used to specify a swarm robotics system due to its capability to model concurrent processes. However, the paper focuses on using the swarm specification to analyse the emergent behaviour and does not address the control design.

In [92] probabilistic temporal verification is applied in an ant-based swarming scenario involving Micro Aerial Vehicles (MAVs) initially proposed by [93]. Micro Aerial Vehi-

[2]The concept of supervisors will be present later in the text.

cles are deployed, from a base, to search for a target user at an unknown location and establish a communication network with the user. The MAVs are positionless, meaning that they are not aware of their position coordinates, and therefore they depend on proprioceptive sensors and local neighbourhood communication. The MAVs position themselves in Y-junction grid that covers a space in the form of an equilateral triangle. Valid positions are located at the border of an MAV's communication range with a neighbour, which is $100$ m. The MAVs move at a speed of $10$ m/s and are deployed one by one with an interval that varies from $7.5$ s to $22.5$ s. They navigate through the grid, choosing between two alternative paths in each node by using a probability that takes into account the amount of deposited pheromone and a constant that determines the attractiveness of unexplored paths. Originally this problem was simulated with $50$ ms intervals by [93]. In [93] it is proposed some abstractions and assumptions to reduce the search space for the model checking.

- Each step is equal to $10$ s;
- At each step, an MAV is deployed;
- The MAV that returns to base only resumes the exploration if there is land signal;
- It is assumed that there will be no collisions.

With these assumptions, one point that is missed out is the case where an MAV is deployed while the previous one did not reach the next node, as the deployment interval and the time to travel from one position to another are the same: $10$ s. These assumptions simplify the problem but are necessary to reduce the search space to a tractable level.

#### 2.1.2.1 Hybrid system theory

Most of the aforementioned methods assume 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 [94, 95, 96, 97, 98, 99, 89, 100, 101, 102].

In particular, hybrid automata can capture continuous and discrete state variables [94]. In [94], the authors explore an analogy between robotic agents and cells in terms of

sensors and actuators. The control states are defined over a discrete set with the time-dependent motion being defined in a continuous space using hybrid automata.

[95] overviews the problems design and verification using HST for safe conflict resolution in the context of distributed air traffic automation. [96] uses HST to control the formation of a group of three robots moving along a given trajectory. Here the discretization is at the controller level: the robot has multiple continuous motion controllers. Its sensory input—in which other robots are perceived—is used to select the controller to be executed.

In graph grammars [103] the symbols of the grammar are graphs, in which vertices represent robots, and the edges represent the connections of two robots. Graph grammars can be extended to be applied in the context of HST. In [97, 98] embedded graph grammars includes the geometry, continuous dynamics, and local condition of systems. Embedded graph grammars are then used for the problem of organising the robots into subgroups while maintaining the overall connectivity.

Modelling large groups of robots can pose a challenge as the combinatorial explosion in the number of states of the system can be untreatable. To avoid this problem a stochastic approach can be assumed, in which the distribution of agents over the state space is modelled [99]. In [99], a probability density function is used to formally define the state of each agent in a large population of robots. The authors apply the minimum principle [104] for the optimal control. This uses partial differential equations to solve the problem of maximising the probability of the presence of robots in a predefined region.

[89] uses 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.

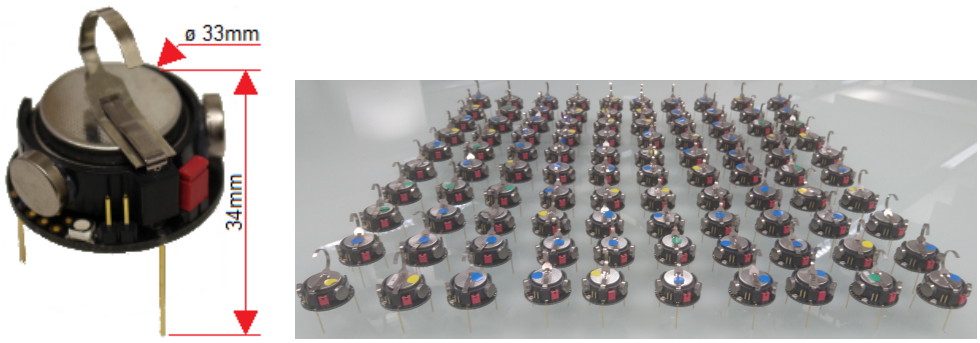
[100] address the problem of maintaining connectivity and speed synchronisation in a network of mobile agents using hybrid control techniques.

In [101] the control of the spatial distribution of mobile agents is modelled as hybrid systems where discrete variables of a Markov process depend on continuous variables. Another example of HST is to control the probability densities of a process with a fixed but unknown state function [102]. This has several potential applications in mobile robotics such as the deployment of robots in an environment based on sensor measurements, search, and monitoring.

### 2.1.3 Platforms

Several swarm robotics platforms are available for use. Two miniature mobile robotics systems are of interest in this work, the Kilobot [38] and the e-puck [37].

#### 2.1.3.1 Kilobot




(a) (b)

Figure 2.1: (a) The Kilobot swarm robot and (b) a swarm of 100 Kilobots.

The Kilobots [38] are designed to make it possible to test collective algorithms on a large group of miniature robots. They provide a low-cost system accessible to robotics researchers. The Kilobot system can be seen in Figure 2.1.

In Figure 2.2 the schematic of the Kilobot is shown highlighting its main components. The Kilobots are equipped with [38]:

- two motors for locomotion based on vibration;
- a microcontroller unit;

- an infra-red (IR) receiver and emitter used for local communication and distance sensing;

- a light sensor.

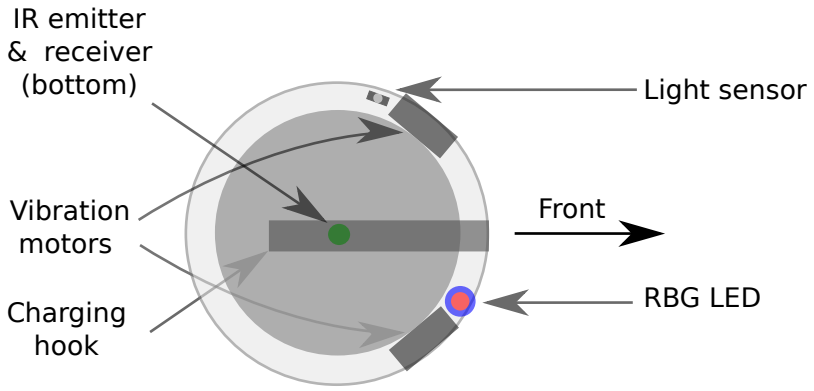


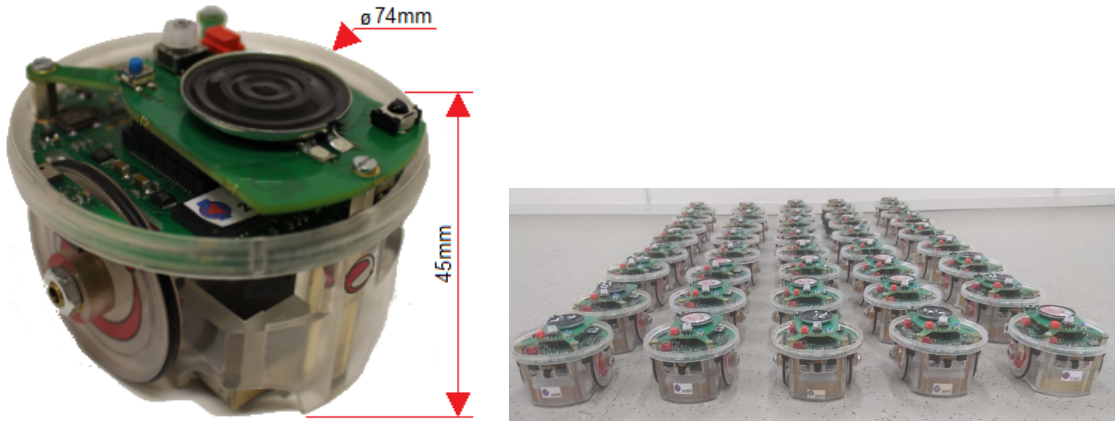

Figure 2.2: The Kilobot's schematic of the position of main components.

The Kilobot platform has been gaining in popularity for research involving many physical robots. In [105] self-assembly in an over-a-thousand-robot swarm is reported. The design of such a large physical system and algorithms for the control of this system is an engineering challenge. The authors demonstrate a swarm of 1024 Kilobots which performed the self-assembly of complex two-dimensional shapes using only local interactions and local sensing. The Kilobots received the desired shape in the form of a binary image. The binary image was captured by the Kilobots using their light sensor, robots then move into the position to additively form the desired shape.

In [106] Kilobots are used for shape formation by self-disassembling. Robots are grouped together and receive the desired shape. They identify whether their location, which is calculated distributively at run-time, belongs to the desired shape. Assisted by light sources positioned at specific locations the robots that do not belong to the shape start to move away. The result is the formation of the desired shape.

#### 2.1.3.2 e-puck

The e-puck robot [37] was designed to meet educational needs at the university level. However, many scholars apply this a platform in their research. The e-puck platform can be seen in Figure 2.3.




(a) (b)

Figure 2.3: (a) The e-puck robot and (b) a swarm of 40 e-pucks.

In Figure 2.4 the schematic of the e-puck is shown highlighting its main components. The e-puck is equipped with [37]:

- sensors: audio, visual, distances to objects, and gravity;
- actuators: two stepper motors and speakers;
- communication: wired and wireless devices;
- microcontroller.

The e-puck has been used for research in several works to investigate a range of swarm strategies. In [107] the segregation of physical e-puck robots based on the Brazil nut effect is experimentally demonstrated. When vibration is applied to a mixture of particles, the largest ones are likely to be on top. In packs of cereal, these are the Brazil nuts, giving the name to this effect. Robots can exhibit the same segregation patterns, based on different virtual attributes, mimicking the behaviour of such particles.

In [52, 53] a strategy that allows a group of e-puck robots to cluster together is presented. The e-pucks can perform a self-organised aggregation with minimal computation. The speed and direction of the e-puck's solely depends on the reading value of a line-of-sight sensor implemented using the robot's camera. The input to output mapping is obtained off-line using grid search.

The cooperative transportation of objects using a swarm of e-puck robots has been studied too [108, 34]. The proposed strategy aims to transport a tall—and possibly heavy—

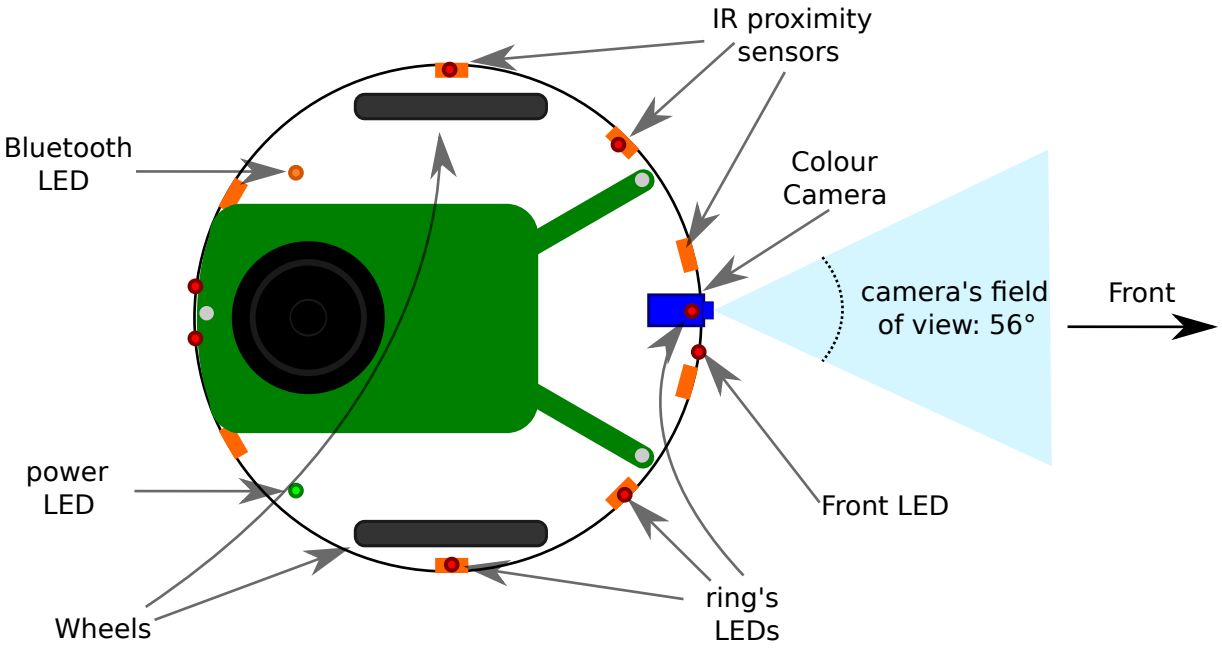



Figure 2.4: The E-puck's schematic of the position of main components.

object to a goal position in a fully distributed manner without relying on communication. Robots can push the object to move it. The resulting movement of the object is guaranteed to converge towards a goal location, as long as the object is of convex shape. Also, e-pucks have been used to validate decentralised segregation strategies applied for the navigation of swarm robotics [109].

In [49] a swarm of five e-puck robots is used to cluster objects initially spread in the environment. As in [52, 53] the e-puck maps the line-of-sight sensor directly onto the speed and direction of its wheels. The mapping is obtained off-line, using genetic algorithms.

The e-puck has also been considered as a standard robot for simulation. Simulations of swarm systems allow the investigation of large swarms while saving time and avoiding the cost of such a system in exchange for the loss of accurate prediction of physical behaviours. The Enki simulation toolkit [110] is an example of a 2D simulator that provides physics and dynamics simulation of swarm robotics systems and faster than real time experiments. For example, in [111] the synchronisation in swarms of mobile agents are investigated. This work shows the correlation between the synchronisation regime and factors such as the agent's speed, angle, and range of interaction.

Algorithms implemented in physical e-puck robots assist the human management of and interaction with a swarm robotics system. In [112] a human operator can interact with a swarm through a management software package capable of organising the swarm via a leader. The operator can request the count of robots and assign robots to workgroups to perform different tasks.

Another example is the use of e-pucks in Enki to study human-robot integration in the context of swarm robotics. In [113] the collaboration between a swarm of virtual robots and a user with limited situational awareness is investigated. Use of more recent devices, such as the Google Glass, in human–robot swarm interaction, has also been considered [114].

The intensive use of the e-puck platform has called attention to some of its limitations. Considering newer platforms, such as the Kilobot, the e-puck is comparatively outdated, with revisions being restricted to replace deprecated hardware components. Nevertheless, it is a platform that is largely available in research centres, and therefore some effort has been made to ensure e-puck's usability. The use of the libIrcom [115, 116] allows the e-puck to perform local communication and detect the distance to nearby robots using the IR sensors, in an approach similar to the one used by the Kilobot. The e-puck when libIrcom, can in addition, also detect the direction of nearby robots.

Another effort to increase the e-puck usability is the development of an operating system for this platform. Open Swarm [117] provides an easy-to-use event-driven interface with the e-puck hardware. Although Open Swarm aims to be deployed in any miniature robots, it has so far only been developed for the e-puck platform.

## 2.2 Languages, grammars, and automata applied to discrete event systems

A language $L$ is defined as a set of words (also called string) $w$ over an alphabet $\Sigma$, where the alphabet is a finite set of symbols. An empty word, $\varepsilon$, is a string with no symbols. The number of symbols in a word $w$, denoted by $|w|$, is called the length of the word. The set of all possible words of an alphabet is $\Sigma^*$ and it is obtained by concatenating zero or more symbols from $\Sigma$. The set of all non-empty word of an alphabet is $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. A language $L$ over an alphabet $\Sigma$ is a subset of $\Sigma^*$ [118].

Let us consider two words $w_1$ and $w_2$. The concatenation of two words $w_1 w_2$ is the juxtaposing of their symbols. The concatenation of two languages $L_1$ and $L_2$ is [118, 119]:

$$L_1 L_2 = \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}. \tag{2.2}$$

The power of a language, $L^k$, is defined as $k$ concatenations of the language, with $L^0 = \{\varepsilon\}$. $L^k$ can be recursively defined as:

$$L^i = \{w_1 w_2 : w_1 \in L^{i-1}, w_2 \in L\} \; \forall i > 0 \tag{2.3}$$

For example, $L^1 = L$, $L^2 = LL$, and $L^3 = LLL$. The complement of a language is all the words that do not belong to the language [118, 119], that is:

$$L' = \Sigma^* - L. \tag{2.4}$$

The *Kleene closure* of a language is [119, 118]:

$$L^* = \bigcup_{i=0}^{\infty} L^i, \tag{2.5}$$

and the positive closure is [118]:

$$L^+ = \bigcup_{i=1}^{\infty} L^i. \tag{2.6}$$

The *prefix-closure* of a language is defined as [119]:

$$\overline{L} = \{s \in \Sigma^* : \exists t \in \Sigma^* \wedge st \in L\}. \tag{2.7}$$

The events of a discrete event system (DES) are related to the symbols from an alphabet of a language [119], and the words formed by these symbols represent sequences of operations (e.g., related to the sensing and actuation). In the context of this thesis, we

consider the use of languages to control robotic systems modelled as DES. The control objective is to guarantee that at any time only valid words or prefix of valid words occur. The problem is that for the majority of real-world application the number of valid words will be infinite or huge. It may be impossible to keep a dictionary of all valid words. So, a representation of the language is needed.

A language representation may be able to (1) recognise words, given a word the formal representation will accept or reject the word as a member of the language; (2) generate words, the representation will generate words that belong to the language or; (3) realise a language, the representation will both recognise and generate words.

Languages can be represented by grammars. A grammar $M$ is a quadruple [118]:

$$M = (V, T, F, P), \tag{2.8}$$

where,

- $V$: is a finite set of variables, also called non-terminal symbols;
- $T$: is a finite set of terminal symbols (the $\Sigma$ of a language);
- $F \in V$: is the start symbol;
- $P$: is a finite set of productions (also called rules). In the form:
  - $< left > \rightarrow < right >$.

A grammar generates words based on an unspecified number of derivations starting in $F$, which may be zero ($\overset{*}{\Longrightarrow}$) or non-zero ($\overset{+}{\Longrightarrow}$) derivations. Thus, a language $L$ is realised by a grammar $M$ as [118]:

$$L(M) = \{w \in T^* : F \overset{*}{\Longrightarrow} w\} \text{ or } L(M) = \{w \in T^* : F \overset{+}{\Longrightarrow} w\}. \tag{2.9}$$

The Chomsky hierarchy [120, 121] classify languages in for classes:

- Type-3 or regular languages [122, pg 31] are generated by productions of one of the forms $A \to Ba$ (left regular) or $A \to aB$ (right regular) but never both in the entire grammar, where $A \in V$ and $B \in V$ are variables and $a \in T$ is a terminal. Also, the forms $A \to B$ and $A \to \varepsilon$ are allowed in both cases. The operator | can be used to specify multiples productions that have the same start, as $A \to aB|a$. The use of left regular and right regular leads to equivalent languages;

- Type-2 or context-free languages [122, pg 100] are generated by productions in the form $A \to \alpha$, where $A \in V$ is a variable and $\alpha$ is a string of terminals ($T$) and variables ($V$);

- Type-1 or context-sensitive languages [118, pg 153] are generated by productions of the form $\alpha A\beta \to \alpha\gamma\beta$, where $A \in V$ is a variable and, $\alpha$, $\beta$ and $\gamma$ are a string of terminals ($T$) and variables ($V$) where $\gamma$ cannot be empty;

- Type-0, unrestricted languages or recursively enumerable languages are the languages recognised by a Turing machine [122, pg 137].

### 2.2.1 Regular languages

The Type-2 languages are recognised by Push-down automata [122, pg 109], in which the non-determinism is necessary. The language class of interest of this work for control is the regular languages, which can be recognised by deterministic finite automata (DFA).

A DFA is a formal machine composed of a set of states, a set of symbols called alphabet, a transition function, an initial state and a set of final states. The machine evolves to others states based on an input given by a tape; when a symbol is read from the tape, the new state is given according to the complete transition functions, which takes into account the current states and the symbol. Thus, the DFA accept a word in the input tape if after it consumes all the input the resulting state of the DFA is part of the set of final states. A formal representation for a DFA, $A$, is the quintuple:

$$A = \{Q, \Sigma, \delta, q_0, Q_m\} \tag{2.10}$$

where,

- $Q$: is a finite set of states;
- $\Sigma$: is a finite set of symbols;
- $\delta : Q \times \Sigma \rightarrow Q$: is the total 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 $Q_m \subseteq Q$.

Another representation for regular languages is a regular expression (RE) (see [122, pg 63]). For example, the RE $a|b$ is the language $L = \{a\} \cup \{b\} = \{a, b\}$. The RE $a^*$ means the language $L = \{a\}^* = \{\varepsilon, a, aa, aaa, ...\}$, that is, the language with any natural number including 0 of $a$, $a^+$ is $a^* - \{\varepsilon\}$. The concatenation is implicit in RE and the parenthesis is used to group parts of the RE that must be evaluated first. For example, $(a|b)b$ is the language $L = ab, bb$, also the operators $*$ or $+$ can be applied to a group. In the RE $(a|b)^*$, the language is $L = \{\varepsilon, a, b, aa, ab, bb, aaa, aab, ...\}$. Let us consider examples of languages over an alphabet $\Sigma = \{a, b, c\}$ defined as:

- zero or more symbols $a$ followed by a suffix $bc$ is expressed as $a^*bc$;
- one or more symbols $a$ followed by a suffix $bc$ is expressed as $a^+bc$ or $aa^*bc$;
- all words with the prefix $abc$ is expressed as $abc(a|b|c)^*$.

Let us consider the RE $a^*bc$. The grammar that realises this RE is $M = \{\{A, B, C\}, \{a, b, c\}, A, P$, where P is,

$$\begin{aligned} A &\rightarrow aA|B \\ B &\rightarrow bC \\ C &\rightarrow c \end{aligned} \tag{2.11}$$

A DFA that recognises this language is $A = \{\{q_0, q_1, q_2, q_f\}, \{a, b, c\}, \delta, q_0, \{q_2\}\}$, where $q_2$ is the only final state and $\delta$ is given by Table 2.1, where $\lambda$ means that there is no transition; in that case the word is rejected. Figure 2.5 presents the graphical representation of DFA $A$.

Table 2.1: Transition function of a DFA accepting language $a^*bc$.

| Current state | symbol $a$ | $b$ | $c$ |
|---|---|---|---|
| $q_0$ | $q_0$ | $q_1$ | $\lambda$ |
| $q_1$ | $\lambda$ | $\lambda$ | $q_2$ |
| $q_2$ | $\lambda$ | $\lambda$ | $\lambda$ |

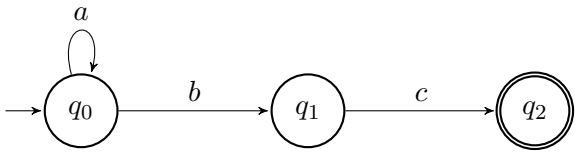



Figure 2.5: An example of a DFA that realises the language $a^*bc$, that is, all the words starting with zero or more symbols $a$ and finishing with the suffix $bc$.

SCT is one of the frameworks to model and synthesise DES using DFA to generate formal languages. The SCT was introduced by [15], and it will be described in more detail in Section 2.3.

### 2.2.2 Stochastic languages

A stochastic language is a probability distribution $D$ over $\Sigma^*$. The probability of a string $x \in \Sigma^*$ under the distribution $D$ is denoted by $Pr_D(x)$, where [123]:

$$\sum_{x \in \Sigma^*} Pr_D(x) = 1 \tag{2.12}$$

Probabilistic finite-state machines (PFSMs) are applied to define the probability distribution $D$ [123]. PFSMs have been applied in a variety of fields, including linguistics, speech and pattern recognition, bioinformatics, circuit testing, analysis of time series, and machine translation. In this section, the concept of probabilistic finite-state automata (PFA) and probabilistic deterministic finite-state automata (PDFA) are presented. The choice for such formal machines allows us to build on the experiences and concepts of the traditional SCT.

#### 2.2.2.1 Probabilistic finite-state automata

In [123] a definition for PFA is given based on the works of [124, 125, 126, 127, 128, 129]. According to [123], a PFA, $A^p$, is defined as:

$$A^p = \{Q, \Sigma, \delta, I, F, P\}, \tag{2.13}$$

where:

- $Q$ is a finite set of states;
- $\Sigma$ is an alphabet;
- $\delta : Q \times \Sigma \times Q$ is the set of transitions;
- $I : Q \to \mathbb{R}^+$ are the initial state probabilities;
- $F : Q \to \mathbb{R}^+$ are the final state probabilities;
- $P : \delta \to \mathbb{R}^+$ are the transition probabilities;

and the sum of all initial state probabilities is equal to $1$, as:

$$\sum_{q \in Q} I(q) = 1. \tag{2.14}$$

Moreover, the probability of a state $q$ being final, $F(q)$, plus the sum of the probabilities of all transitions with origin in state $q$ is equal to $1$, as:

$$\forall q \in Q, F(q) + \sum_{e \in \Sigma, q' \in Q} P(q, e, q') = 1, \tag{2.15}$$

Note we consider that $0 \in \mathbb{R}^+$ and that $P(q, e, q') = 0$ if $\delta(q, e, q')$ is not defined.

In this thesis, PFAs are represented as graphs. The transition $\delta(q, e, q')$—where $q, q' \in Q$ are the origin and target states, respectively and $e \in \Sigma$ denotes the symbol—are represented by labelled arcs. The arcs' label are in the form $e : p$, where $p = P(q, e, q')$ (i.e.,

the associated probability of the transition). States have the label in the form $q : (i, f)$, where $q \in Q$ is the state, $i \in I$ is the probability $I(q)$, and $f \in F$ is the probability $F(q)$. Figure 2.6 shows a PFA with for states $Q = \{q_0, q_1, q_2, q_3\}$ over an alphabet $\Sigma = \{a, b, c\}$. It has two initial states: $q_0$ and $q_2$, that is $I(q_0), I(q_2) > 0$, with $I = \{(q_0, 0.6), (q_2, 0.4)\}$. It has three final states: $q_0$, $q_1$, and $q_3$, that is $F(q_0), F(q_1), F(q_3) > 0$, with $F = \{(q_0, 0.1), (q_1, 1.0), (q_3, 1.0)\}$. The regular expression for the language $L$ recognised by the PFA of Figure 2.6 is $a^*(a|(b|c)c^*b)$.

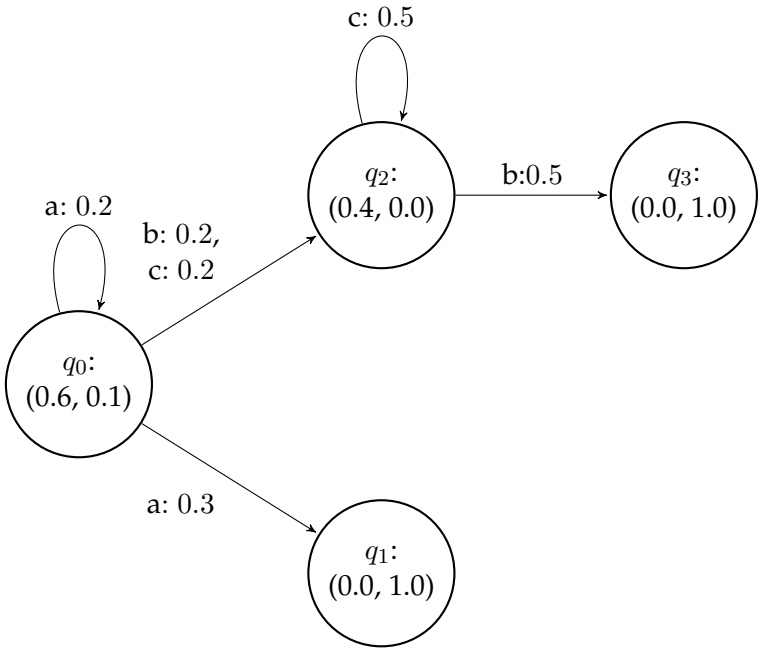



Figure 2.6: An example of a PFA. Arcs represent transitions triggered by a symbol and its associated probability. Nodes represent states. A node's label informs the state name and a pair of probabilities: the probabilities of the state are for the initial state and final state, respectively.

#### 2.2.2.2 Probabilistic deterministic finite-state automata

A PDFA is a sub-class of PFA where [123]:

- $\exists q_0 \in Q : I(q_0) = 1$, that is, there is only a single initial state;
- $\forall q \in Q, \forall e \in \Sigma, |\{q' : (q, e, q') \in \delta\}| \leq 1$, that is, the transition function $\delta$ is defined completely by $q$ and $e$.

Therefore, a PDFA can be denoted by:

$$A^{pd} = \{Q, \Sigma, \delta, q_0, F, P\}, \tag{2.16}$$

where:

- $Q$ is a finite set of states;
- $\Sigma$ is the alphabet;
- $\delta : Q \times \Sigma$ is the set of transitions;
- $q_0 \in Q$ is the initial state;
- $P : \delta \to \mathbb{R}^+$ is the transition probabilities;
- $F : Q \to \mathbb{R}^+$ is the final state probabilities;

further, the probability of a state $q$ be final, $F(q)$, plus the sum of the probabilities of all transitions with origin in state $q$ is equal to 1, as:

$$\forall q \in Q, F(q) + \sum_{e \in \Sigma, q' \in Q} P(q, e, q') = 1, \tag{2.17}$$

In [123] it is proved that PFA and PDFA are not equivalent as the determinism on PFA implies a loss of expressive power.

## 2.3 Supervisory control of discrete event systems

DES is an abstraction for a large variety of problems [18]. DESs are all those systems which can be expressed in a discrete state. In a DES the state space is a discrete set, which means that the change from one state to another happens abruptly. In the DES, the transition function is always relating the occurrence of an event in the current discrete state with another discrete state. That is, changes in state (called transitions) are triggered by events [119].

SCT [15, 16, 17] is a theoretical framework for synthesising controllers, called supervisors. It assumes that the systems under investigation can be represented as discrete event systems (DES). SCT distinguishes the events that drive the evolution of the system between uncontrollable events and controllable events. Uncontrollable events represent control input, such as feedback signals, for example, from sensors. Controllable events represent the control output, such as 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. Each symbol of the language's alphabet 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").

### 2.3.1 Generators

The class of formal languages that is most commonly used in SCT are the regular languages, also called Type-3 languages [120, 121]. The 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 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. As a result the transition function of a generator is partial in contrast to the total transition function used by automata. A generator $G$ is a 5-tuple:

$$G = (Q, \Sigma, \delta, q_0, Q_m), \tag{2.18}$$

where:

- $Q$ is a finite set of states;
- $\Sigma$ is a finite set of symbols related to the system's events;
- $\delta : Q \times \Sigma \to Q$ is a partial transition function;
- $q_0 \in Q$ is the initial state;
- and $Q_m \subseteq Q$ is a set of marked states.

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 ($\Sigma_u$) and controllable events ($\Sigma_c$), 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 $\Sigma^*$ denote the set of all words—or sequences of events—over an alphabet $\Sigma$. Let $\Sigma^+$ denote the set of all words excluding the empty word $\epsilon$ (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.

### 2.3.2 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 [130]). The


(a) $G_1$ (b) $G_2$

Figure 2.7: Examples 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 $Q_i = \{q_1, q_2, ...\}$. $move$ and $stop$ are controllable events. $active$ and $inactive$ are uncontrollable events.

free behaviour modules are realised by generators $G_i,\ \ i \in \{1, 2, \ldots, m\}$. By default, it is assumed that the free behaviour models are independent of each other. Figure 2.7 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.

### 2.3.3 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 $E_j,\ \ j \in \{1, 2, \ldots, n\}$. Figure 2.8 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, 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 Figure 2.8), 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

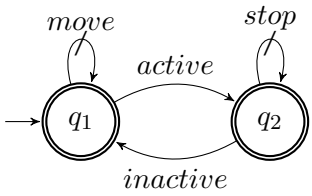



Figure 2.8: An example of a control specification that enables the conveyor to move only when there is no part in front of the sensor.

a state with the buffer empty; thus, only the state that represents the empty buffer is marked.

### 2.3.4 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 achieved by combining the free behaviour models and specifications using synchronous composition.

#### 2.3.4.1 Synchronous composition

The synchronous composition (represented by $\cdot||\cdot$) of two generators $G_a$ and $G_b$ with alphabets $\Sigma_a$ and $\Sigma_a$, respectively, is defined as:

$$G_a||G_b = (Q_a \times Q_b, \Sigma_a \cup \Sigma_b, \delta_{a||b}, (q_{0_a}, q_{0_b}), Q_{m_a} \times Q_{m_b}), \tag{2.19}$$

where

$$\delta_{a||b}((q_a, q_b), e) = \begin{cases} (\delta_a(q_a, e), \delta_b(q_b, e)) & \text{if } \delta_a(q_a, e)! \wedge \delta_b(q_b, e)! \wedge e \in \Sigma_a \wedge e \in \Sigma_b \\ (\delta_a(q_a, e), q_b) & \text{if } \delta_a(q_a, e)! \wedge e \in \Sigma_a \wedge e \notin \Sigma_b \\ (q_a, \delta_b(q_b, e)) & \text{if } \delta_b(q_b, e)! \wedge e \notin \Sigma_a \wedge e \in \Sigma_b \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{2.20}$$

and $\delta(x, y)!$ means that $\delta$ is defined on input $(x, y)$. Equation 2.20 ensures that events that are not common to $\Sigma_a$ and $\Sigma_b$ 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:

$$K = G||E. \tag{2.21}$$

#### 2.3.4.2 Controllability of the target language

It is important to note that the target language is not necessarily controllable. A language $K$ over an alphabet $\Sigma$ is controllable with respect to the free behaviour model $G$ and the set of uncontrollable events $\Sigma_u \subseteq \Sigma$, if [119]:

$$\forall s \in \overline{L(K)}, \forall e_u \in \Sigma_u, se_u \in L(G) \Rightarrow se_u \in \overline{L(K)}, \tag{2.22}$$

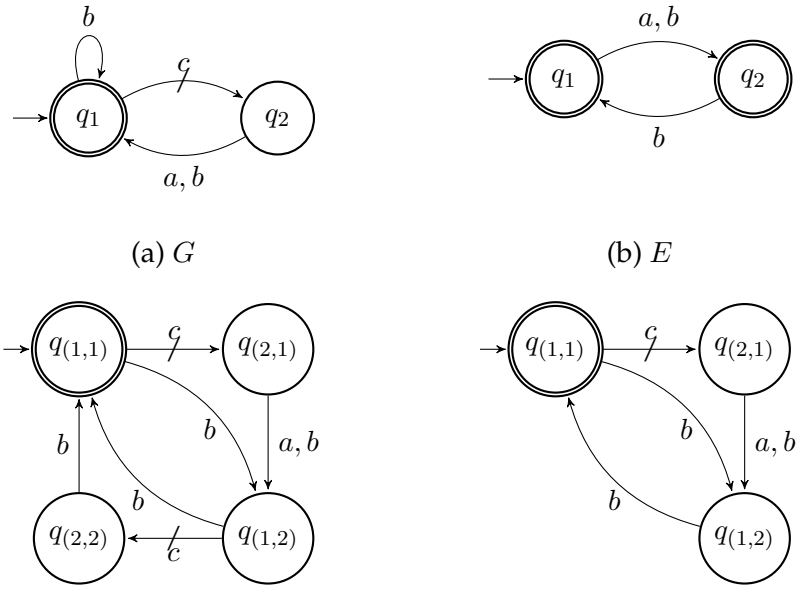
In other words, if $s$ is a prefix of a word of the language generated by $K$, $L(K)$, and $e_u$ an uncontrollable event that is physically possible to occur after this sequence (i.e. $se_u \in L(G)$) , then $se_u$ must also be a prefix of a word in $L(K)$ (i.e. $se_u \in \overline{L(K)}$).

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 Equation 2.20. 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:

$$\exists\, e \in \Sigma_u : e \in \Sigma_{G(x)}\ and\ e \notin \Sigma_{K(y)}, \tag{2.23}$$

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 [15]. In other words, it is the largest sub-language of $K$ that is controllable.

Figure 2.9 shows an example of a bad state and its removal. The composition of $G$, Figure 2.9(a), and $E$, Figure 2.9(b), results in target language $K$, Figure 2.9(c). 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, Figure 2.9(d), is controllable.




(a) $G$ (b) $E$

(c) $K = G||E$ (d)

Figure 2.9: An 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.

#### 2.3.4.3 Accessibility

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.

#### 2.3.4.4 Co-accessibility

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.

#### 2.3.4.5 Trim

All non-accessible and non-coaccessible states of $G$ can be removed with the operator $\mathrm{trim}(G)$.

Figure 2.10 shows an example of a non-coaccessible state and its removal by the $\mathrm{trim}$ operator. This example considers the controllable event $a$ and the uncontrollable events $b$ and $c$. The generator in Figure 2.10(a) 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 Figure 2.10(b)). 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 (Figure 2.10(c)). Note that as only transitions triggered by controllable events led to $q_3$, the resulting target language is controllable.

#### 2.3.4.6 Maximal controllable sub-language

The maximal controllable sub-language of a target language $K$ over a free behaviour model $G$ is obtained by the $SupC(G, K)$ operator. $SupC(G, K)$ removes bad states from language $K$ (obtained by Equation 2.21) and applies the $\mathrm{trim}$ operator to guarantee the

accessibility and co-accessibility of the resulting generator. It works iteratively as the $\mathrm{trim}$ can result in additional bad states. The remaining language,

$$L_m(S/G) = SupC(G, K), \tag{2.24}$$

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 [15], modular [16], and local modular [20, 19, 22] supervisors.

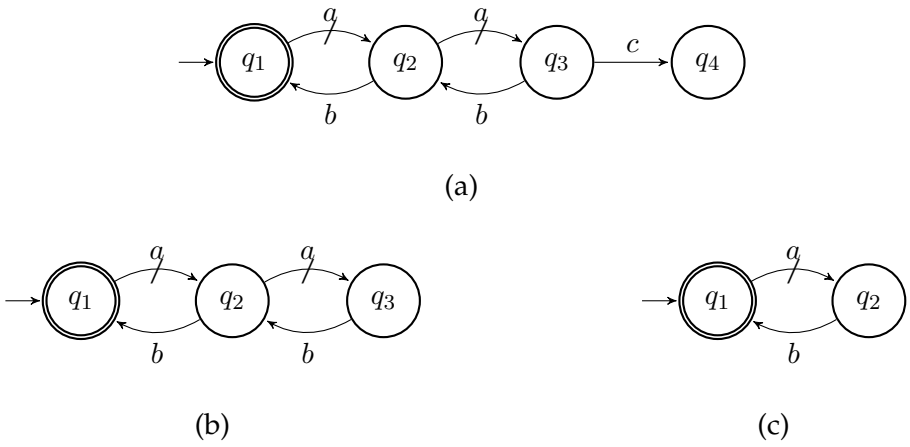



Figure 2.10: An example of a non-coaccessible state and its removal by $\mathrm{trim}(\cdot)$. In (a), states $q_1$, $q_2$, and $q_3$ are coaccessible, as all of them have a path to marked state $q_1$. State $q_4$ is not coaccessible and hence removed (b). As state $q_4$ could be reached from state $q_3$ 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.

#### 2.3.4.7 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 $m$ free behaviour models in a single generator:

$$G = G_1 || \ldots || G_m. \tag{2.25}$$

All $n$ specifications are also composed of a single generator:

$$E = E_1 || \ldots || E_n. \tag{2.26}$$

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

$$K = G||E. \tag{2.27}$$

Finally, the monolithic supervisor $S$ is obtained as:

$$S \ : \ L_m(S/G) = SupC(G, K). \tag{2.28}$$

#### 2.3.4.8 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 [16]. In this approach, the modularity of the specifications is explored. The modular approach composes one supervisor for each specification. These supervisors can then be executed in parallel.

The free behaviour models are composed of a single generator $G^{mod}$. This is done in the same way as for the monolithic approach (see Equation 2.25). Thus, $G^{mod} = G$.

Rather than calculating a single target language, one target language $K_j^{mod}$ is computed for each specification $E_j$:

$$K_j^{mod} = G^{mod}||E_j \quad \forall \ j \in \{1, \ldots, n\}. \tag{2.29}$$

The modular supervisor is obtained for each target language, analogous to Equation 2.28:

$$S_j^{mod} : L_m(S_j^{mod}/G^{mod}) = SupC(G^{mod}, K_j^{mod}) \quad \forall \ j \in \{1, \ldots, n\}. \tag{2.30}$$

The modular approach requires the specifications to have no conflicts. To check for conflicts all modular supervisors are composed together into $S_{||}^{mod}$. $S_{||}^{mod}$ is then compared with the monolithic supervisor. If they are not equivalent, that is, they produce different languages, then a conflict exists (see [16, 1], 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 $K_{1,2}^{mod} = G||E_1||E_2$.

#### 2.3.4.9 Local modular supervisors

The local modular approach [20, 19, 22] 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 $E_j$.

### 2.3.5 Controller implementation

The SCT states that the supervisors in the controller implementation can only disable controllable events and the physical plant is responsible for generating both types of events: controllable and uncontrollable [18, 22]. However, a real controller acts by sensing stimuli from the physical plant and answering with commands. Usually, those stimuli are uncontrollable events and commands are controllable events. Whereas the plant spontaneously generates the uncontrollable events it does not spontaneously generates the controllable events (see causality problem in [18]).

[22] suggests an architecture that can agree with the SCT and with the control paradigm. This architecture is shown in Figure 2.11 and it is based on the inclusion of a product system. The product system is the free behaviour generators and it is included as an

artificial piece of the plant inside the control system responsible to generate the controllable events. Thus, the controllable events are generated inside the control system as required by the traditional control approach, but the supervisor only disables these events as requested by the SCT.

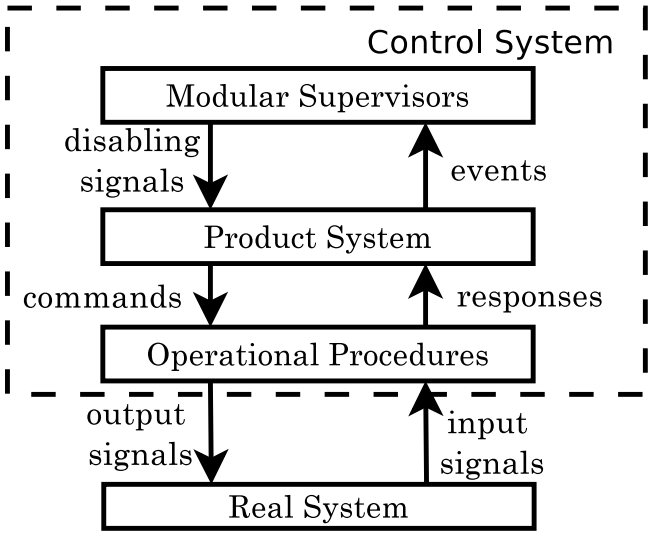



Figure 2.11: Control structure adapted from [22]. The supervisor is the synthesised control. The product system is included to solve the causality problem. The operational procedure is responsible to relate the discrete events with the physical system.

In fact, the supervisors and product systems are a data structure (see Section 4.1.1) that represents the synthesised control logic and free behaviours models, respectively. Based on this data, another module not shown in the architecture, called generator player (see Section 4.1.2), generates the events and controls all the flow of the controller. This discrepancy happens because this architecture was created concerning the programmable control logic (PLC) implementations, which is very common in manufacturing coordination control. In Section 4.1 a implementation focused on swarm robotics is presented.

Furthermore, the architecture states an operational procedure layer (see Section 4.1.3) that links events and commands to the real system. The operational procedure layer requires some programming, but normally the amount of code is several times smaller than the code for the whole control logic.

### 2.3.6 Applications

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 [21, 24, 22, 1]. SCT is also applied to design controllers for systems of multiple robots [131, 31, 32], 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 validation using physical robots. In [31, 32] authors agree that the modular approach used in simulation only is a limiting factor due to the increasing state size of the automata representation. Moreover, the works by [31, 32] are only partially based on the RW framework [15, 17]; 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 [132, 133] and persons, for example, in theme parks [134].

### 2.3.7 Probabilistic generators

In this section we introduce the concept of probabilistic finite-state generators, which for simplicity we will refer as probabilistic generators (PG). Whereas the difference on the formal definition of DFA and generators[3] is the use of partial transition functions for generators and total transition function for DFA, PG and PFA have more substantial differences. First, PG have a single initial state ($q_0$), in contrast with PFA. Second, the PG transition function is deterministic, in contrast with PFA.

As in generators, in PG the symbols, $\Sigma$, are related to discrete events. Therefore, $\Sigma$ is referred to as a set of events. Events are also partitioned into two disjoint sets of controllable and uncontrollable events.

Furthermore, multiple definitions for probabilistic generators have been proposed. The definition of [40] does not contain a definition for final states, differently from PFA and PDFA. A probabilistic generator $G^p$ is defined as [40]:

$$G^p = \{Q, \Sigma, \delta, q_0, p\}, \tag{2.31}$$

[3]For the sake of brevity when the term generator is used it is referring to non-probabilistic generators.

where:

- $Q$ is a finite set of states;
- $\Sigma$ is a finite set of events, with $\Sigma = \Sigma_u \cup \Sigma_c$, where:
  - $\Sigma_u$ is the set of uncontrollable events and;
  - $\Sigma_c$ is the set of controllable events;
- $\delta : Q \times \Sigma \to Q$ is the partial transition function;
- $q_0$ is the initial state, where $q_0 \in Q$.

The probability of an event occurring in a particular state is:

$$p : Q \times \Sigma \to [0, 1], \tag{2.32}$$

where

$$\forall q \in Q, \sum_{e \in \Sigma} p(q, e) \leq 1. \tag{2.33}$$

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 (i.e., if $\exists q \in Q, \sum_{e \in \Sigma} p(q, e) < 1$) [40].

A different definition for probabilistic generators, given by [39], included marked states, $Q_m$:

$$G^p = \{Q, \Sigma, \delta, q_0, Q_m, p\}. \tag{2.34}$$

## 2.4 Summary

Swarm robotics systems are composed of a larger number of robots that are interacting and cooperating to achieve certain goals. Swarms of robots have the potential to

be applied to real-world scenarios and tackle many interesting problems. The design of swarm robotics systems may be categorised into automatic design or behaviour-based design methods, though these classes may also overlap. The modelling of swarm robotics systems may be categorised by sensor-based, microscopic, macroscopic, and cellular automata; though sensor-based and cellular automata can be see as subcategories of the other categories. The control of swarm robotics systems may be categorised by centralised and decentralised; the latter being separated in distributed and hierarchical. This work focuses on the use of microscopic methods, that could be obtain using behaviour-based or automatic design techniques, controlled by a distributed decentralised implementation.

For much of present swarm robotics systems the controllers are still developed in an ad-hoc manner, hindering the transition of swarm robotics to real-world applications. The use of formal approaches could greatly benefit the development. In this chapter, related works that use formal methods to model, analyse, and synthesise swarm robotic controllers were reviewed.

This chapter overviewed the theory of formal languages with a focus on regular languages. The use of regular languages could enable a smooth transition from the ad-hoc development as regular languages can be realised by state machines, which are already widely often employed by the swarm robotics community. Formal languages can recognise or generates words that belong to a language. Each symbol of the alphabet of a particular language can be related to a discrete event in the swarm robotics system (e.g., events related to sensing and actuation). Thus, a controller can be obtained as a language that contains only valid sequences of events. As the control specification and the system model is represented by formal languages, it facilitates the application of formal analysis to guarantee the correctness of the control.

While some works address the use of formal approaches in swarm robotics, there is a lack of work addressing automatic code generation. The SCT provides a framework that can be adapted to automatically produce the source code to control swarms of robots.

# 3

# Design and synthesis of supervisors for controlling swarms of robots

In this chapter it is discussed how to model the capabilities, how to specify the desired behaviour, and how to synthesise controllers in the form of supervisors for swarm robotics systems using the supervisory control theory (SCT) framework.

The method to obtain a supervisor consists of three steps:

1. The capabilities of the robot must be formally defined using generators, called free behaviour models. In general, each component of the robot (e.g., a wheel, a camera, etc.) is modelled individually. Examples are provided in Section 3.1.

2. The desired behaviour of the robot is defined using generators, called specifications. Specifications restrict the free behaviour of the robots. Each specification can be modular, that is, it can consider a single aspect involving a subset of the free behaviour models. Examples are provided in Section 3.1.

3. The free behaviour models and specifications are combined into a supervisor in a process called synthesis. In this process, a generator that realises a target language is obtained through the synchronous composition of free behaviour models and specifications. This generator is further modified by the iterative removal of states that would cause the supervisor to be non-admissible. Three state-of-art synthesis methods are demonstrated in Section 3.2.

## 3.1 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 five case studies that illustrate how SCT can be applied in swarm robotics. The case studies make use of two robotic platforms, the Kilobot [38] and the e-puck [37]. Both platforms move on the ground and are able to locally broadcast messages.[1]

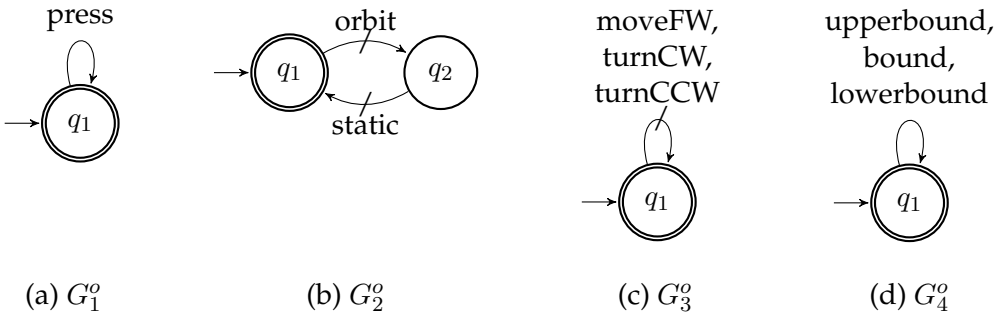
Two case studies—*orbit* and *segregation*—use both robotic platforms. This shows that as long as all task-relevant hardware is available, the same supervisors can be applied to different robotic platforms. Two further case studies, using the e-puck platform, illustrate how state-of-the-art solutions for the problems of *aggregation* [53] and *object clustering* [49] 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 $\theta \in \{o, s, a, c, f\}$ to refer to the different case studies, where $o$ refers to to orbit, $s$ to segregation, $a$ to aggregation, $c$ to object clustering, and $f$ to group formation. We represent the number of free behaviour models by $m$ and the number of specifications by $n$. $G_i^\theta$ denotes the $i$th of $m^\theta$ free behaviour models and $E_j^\theta$ denotes the $j$th of $n^\theta$ control specifications.

### 3.1.1 Orbit

The first case study involves two robots. One robot is static and the other orbits around it [38]. The static robot broadcasts periodically an infrared (IR) message. The orbiting robot uses the IR message to estimate its distance to the static robot. It moves counter-clockwise (CCW) around the static robot. In particular, it modulates its behaviour according to its distance (using two thresholds): (i) if the distance is smaller than the lower threshold, the robot turns clockwise (CW); (ii) if the distance is bigger than the upper threshold, the robot turns counter-clockwise (CCW); and (iii) if the robot is inside the boundary defined by the thresholds it moves forward. Note that the robot is assumed to be within the boundary at the start.

[1] The e-puck requires a non-standard library to broadcast messages using infrared.




(a) $G_1^o$ (b) $G_2^o$ (c) $G_3^o$ (d) $G_4^o$

Figure 3.1: Free behaviour models for the orbit case study. (a) Input device to configure the robot; (b) the robot's ability to assume the roles of a static robot or an orbiting robot; (c) motion capabilities; (d) boundary sensor. Marked states ($Q_m$) are indicated by double lines.

Figure 3.1 shows the free behaviour models, $G_1^o$, $G_2^o$, $G_3^o$, and $G_4^o$. In Table 3.1 the events' definition is summarised. $G_1^o$ represents a user input device that triggers the uncontrollable event press. It is used to configure the robot. $G_2^o$ represents the robot's ability to assume the roles of an orbiting robot (state $q_1$) and a static robot (state $q_2$). Note that each $G_i^o$ has its own set of states $Q_i = \{q_1, q_2, \ldots\}$. $G_3^o$ represents the motion capabilities of the robot. The robot executes one of three movements—move forward, turn CW, and turn CCW—until a new command is issued. The movements are triggered by controllable events moveFW, turnCW, and turnCCW. $G_4^o$ represents a boundary sensor, which measures the distance between the robots and generates uncontrollable events upperbound (too far), bound (within boundaries), and lowerbound (too close). In intervals of $0.2$ s one event is triggered. Due to the uncertainty of the distance measurement, (as a result of noise in the IR sensor), these uncontrollable events can occur in any order (e.g. directly from upperbound to lowerbound).

The orbit behaviour is defined by three specifications, $E_1^o$, $E_2^o$, and $E_3^o$, illustrated in Figure 3.2. $E_1^o$ specifies the configuration of the robot through user interaction. When press occurs the robot shall configure its role; it then switches from static to orbit, or vice versa (as a result of free behaviour model $G_2^o$). $E_2^o$ allows the robot to move only when configured as an orbiting robot. The main strategy is defined by $E_3^o$ where the motion of the orbiting robot is specified by the states $q_1$ (too far), $q_2$ (within boundaries), and $q_3$ (too close). The robot is placed within the boundaries at the start of the experiment and, consequently, the initial state is $q_2$.

Table 3.1: Summary of events' definition for the orbit strategy. In the controllability column controllable events are indicated by *C* and uncontrollable events are indicated by *U*.

| event | controllability | definition |
|---|---|---|
| press | *U* | Detection of the input device actuation. |
| static | *C* | Robot assumes the role of a static robot. |
| orbit | *C* | Robot assumes the role of an orbiting robot. |
| moveFW | *C* | Robot starts to move forward and continues doing so indefinitely. |
| turnCW | *C* | Robot starts to turn clockwise and continues doing so indefinitely. |
| turnCCW | *C* | Robot starts to turn counter-clockwise and continues doing so indefinitely. |
| upperbound | *U* | Orbiting robot is too far from the static one. |
| bound | *U* | Orbiting robot is within the defined distance to the static one. |
| lowerbound | *U* | Orbiting robot is too close from the static one. |

### 3.1.2 Segregation

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 [55]. 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 Figure 3.3. In Table 3.2 the events' definition is summarised. $G_1^s$ represents a user input device that triggers the uncontrollable event *press*. It is used to configure the robot. $G_2^s$ defines the robot 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

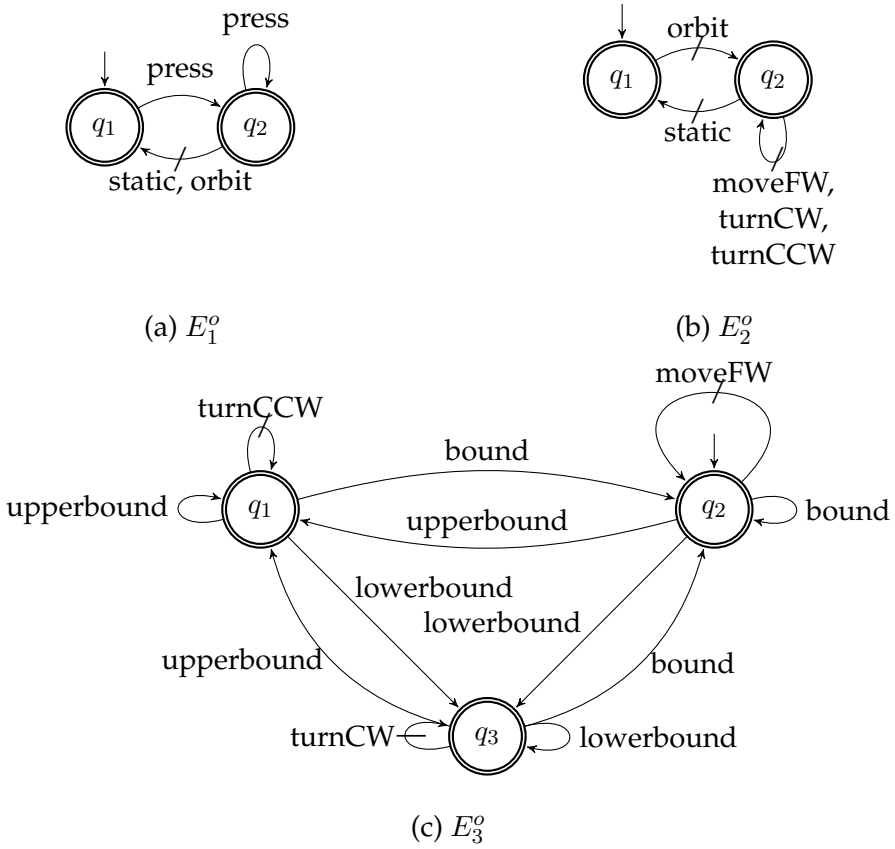
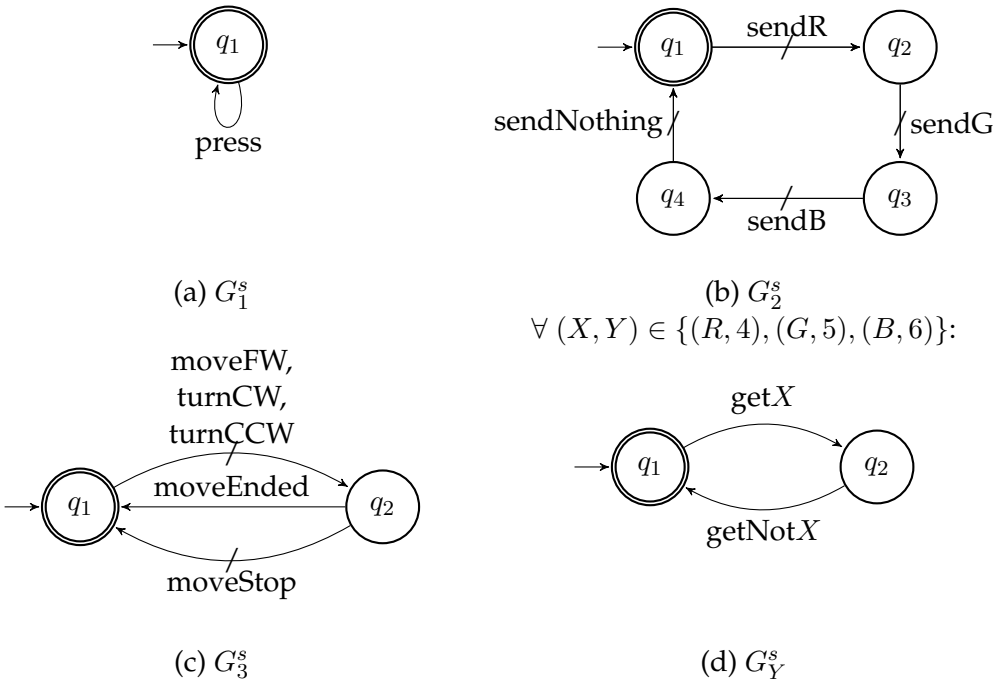



Figure 3.2: Specification for the orbit case study. Each specification, $E_j^o$, has its own set of states $Q_j = \{q_1, ..., q_m\}$. (a) Configures the robot through user interaction as either static or orbiting; (b) prevents the static robot from moving; (c) moves the orbiting robot according to its distance to the static robot.

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.

Figure 3.4 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-




(a) $G_1^s$

(b) $G_2^s$
$\forall\,(X,Y) \in \{(R,4),(G,5),(B,6)\}$:

(c) $G_3^s$

(d) $G_Y^s$

Figure 3.3: 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.

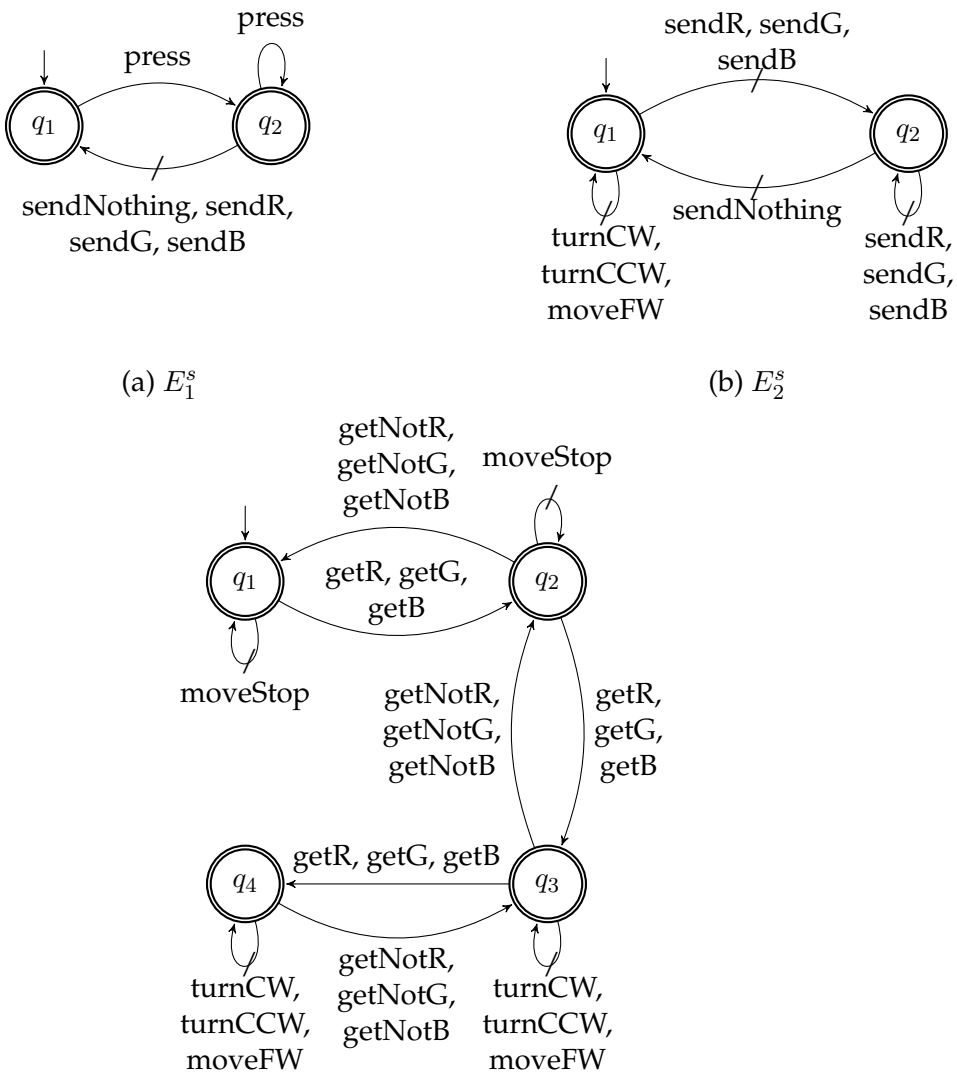
ure 3.3(b)), the robot type is restricted to change sequentially through states of follower ($q_1$), red leader ($q_2$), green leader ($q_3$), and blue leader ($q_4$). Specification $E_2^s$ allows only followers to move. It also sets up the message broadcasting of leaders. The main strategy is represented 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 [18]). 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 Figure 3.3(d)); the same is true for *getG/getNotG* as well as for *getB/getNotB*. This property is exploited in control specification $E_3^s$.

Table 3.2: Summary of events' definition for the segregation strategy. In the controllability column controllable events are indicated by *C* and uncontrollable events are indicated by *U*.

| event | controllability | definition |
|---|---|---|
| press | *U* | Detection of the input device actuation. |
| sendR | *C* | Robot assumes the role of a red leader and starts to broadcast the respective message. |
| sendG | *C* | Robot assumes the role of a green leader and starts to broadcast the respective message. |
| sendB | *C* | Robot assumes the role of a blue leader and starts to broadcast the respective message. |
| sendNothing | *C* | Robot assumes the role of a follower and stops broadcasting any message. |
| moveFW | *C* | Robot starts to move forward and continues doing so for a random period of time. |
| turnCW | *C* | Robot starts to turn clockwise and continues doing so for a random period of time. |
| turnCCW | *C* | Robot starts to turn counter-clockwise and continues doing so for a random period of time. |
| moveEnded | *U* | Signals that the last movement's period has ended and the robot stopped. |
| moveStop | *C* | Stops the robot (the controller forces the robot to stop ignoring any time left). |
| getR | *U* | Robot got a message from a red leader in the past $0.2$ s. |
| getNotR | *U* | Robot did not get a message from a red leader in the past $0.2$ s. |
| getG | *U* | Robot got a message from a green leader in the past $0.2$ s. |
| getNotG | *U* | Robot did not get a message from a green leader in the past $0.2$ s. |
| getB | *U* | Robot got a message from a blue leader in the past $0.2$ s. |
| getNotB | *U* | Robot did not get a message from a blue leader in the past $0.2$ s. |

### 3.1.3 Aggregation

The aggregation strategy allows a group of e-puck robots to gather in a homogeneous environment [53]. 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. 53 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



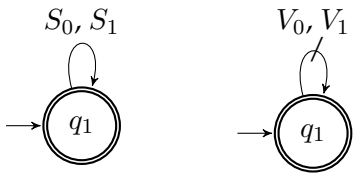

(a) $E_1^s$ (b) $E_2^s$

(c) $E_3^s$

Figure 3.4: 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.

robots [53]. In the following, we show how to formalise this controller using SCT.

Figure 3.5 shows the free behaviour models for the aggregation strategy. In Table 3.3 the events' definition is summarised. Free behaviour model $G_1^a$ represents the binary sensor; uncontrollable events $S_0$ and $S_1$ represent sensor readings $I = 0$ and $I = 1$, re-




(a) $G_1^a$ (b) $G_2^a$

Figure 3.5: Free behaviour models for the aggregation case study. (a) Binary sensor; (b) motion capabilities.

Table 3.3: Summary of events' definition for the aggregation strategy. In the controllability column controllable events are indicated by *C* and uncontrollable events are indicated by *U*.

| event | controllability | definition |
|---|---|---|
| $S_0$ | *U* | Binary sensor detects wall (or nothing) in its line-of-sight. |
| $S_1$ | *U* | Binary sensor detects a robot in its line-of-sight. |
| $V_0$ | *C* | Robot's wheels speeds are set to $(v_{l0}, v_{r0}) = (-0.7, -1)$. |
| $V_1$ | *C* | Robot's wheels speeds are set to $(v_{l1}, v_{r1}) = (1, -1)$. |

spectively. 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 3.6 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.

Instead of specifications $E_3^a$ and $E_4^a$, one may apply one of the specifications shown in Figure 3.7. Note that the specification shown in Figure 3.7(a) 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 Figure 3.7(b) forces $V_1$ to occur first. The third option, Figure 3.7(c), is however equivalent to $E_3^a$ and $E_4^a$; it can be obtained by their synchronous composition (for details, see Section 2.3.4).

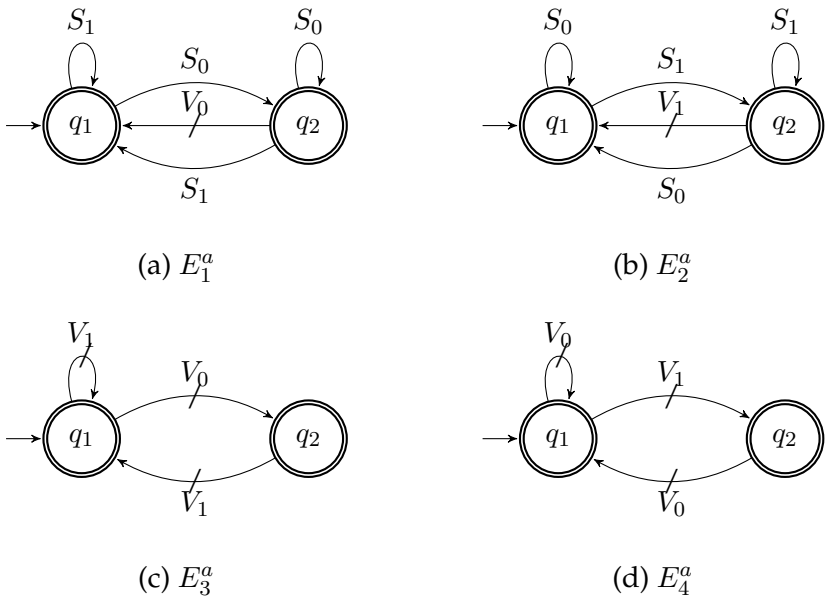



Figure 3.6: Specification for the aggregation case study. (a-b) Moves the robot according to the sensor reading; (c-d) prevents the same movement event from occurring consecutively (yet, the robot will perform its current movement indefinitely).

### 3.1.4 Object clustering

The object clustering strategy allows a group of e-puck robots to cluster objects that are initially dispersed in the environment [49]. Each robot can detect the presence 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 [49]:

$$x = (v_{l0}, v_{r0}, v_{l1}, v_{r1}, v_{l2}, v_{r2}) = (0.5, 1, 1, 0.5, 0.1, 0.5), \tag{3.1}$$

where, $v_{lI}$ and $v_{rI}$ are the left and right wheel velocities, respectively, when the sensor

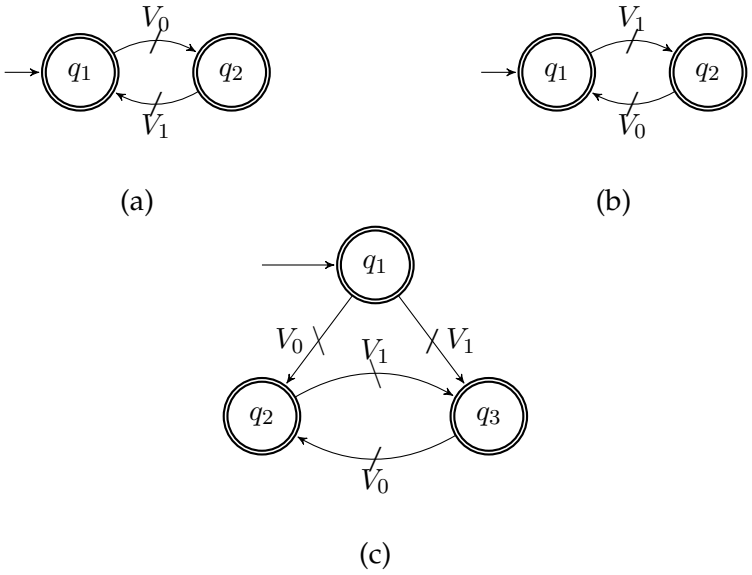



Figure 3.7: Three alternative specifications to be used instead of $E_3^a$ and $E_4^a$ in the aggregation case study. The specifications in (a) or (b) are not fully equivalent to $E_3^a$ and $E_4^a$, as they restrict the first movement to be either $V_0$ or $V_1$, respectively. The specification in (c) is the result of synchronous composition $E_3^a||E_4^a$, and thus fully equivalent.

reading is $I$.

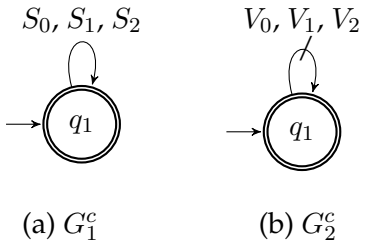
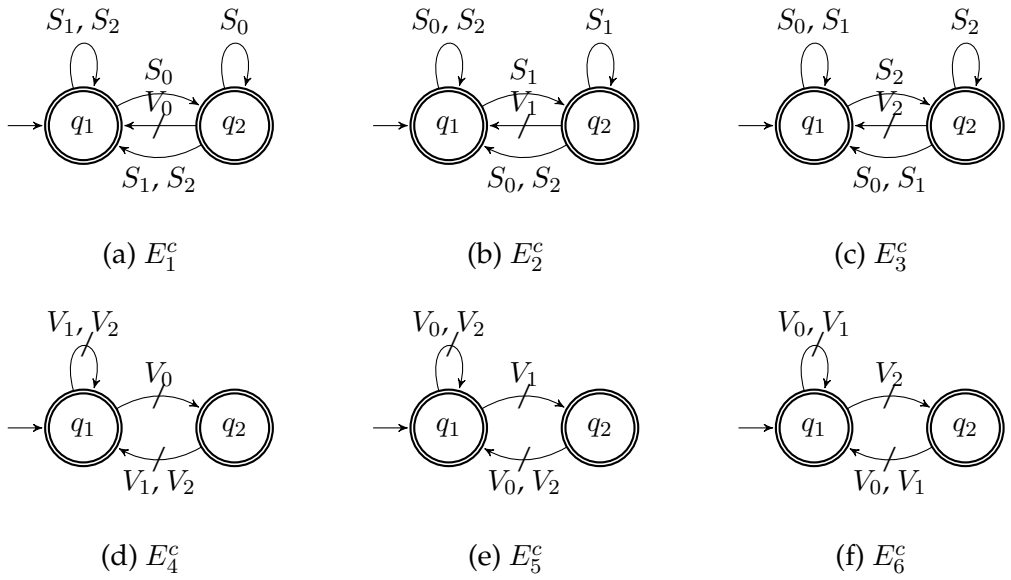



Figure 3.8: Free behaviour models for the object clustering case study. (a) Sensor to detect nothing, objects, or robots; (b) motion capabilities.

Figure 3.8 shows the free behaviour models for this strategy. In Table 3.4 the events' definition is summarised. 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.

Table 3.4: Summary of events' definition for the object clustering strategy. In the controllability column controllable events are indicated by *C* and uncontrollable events are indicated by *U*.

| event | controllability | definition |
|---|---|---|
| $S_0$ | *U* | Binary sensor detects wall (or nothing) in its line-of-sight. |
| $S_1$ | *U* | Binary sensor detects an object in its line-of-sight. |
| $S_2$ | *U* | Binary sensor detects a robot in its line-of-sight. |
| $V_0$ | *C* | Robot's wheels speeds are set to $(v_{l0}, v_{r0}) = (0.5, 1)$. |
| $V_1$ | *C* | Robot's wheels speeds are set to $(v_{l1}, v_{r1}) = (1, 0.5)$. |
| $V_2$ | *C* | Robot's wheels speeds are set to $(v_{l1}, v_{r1}) = (0.1, 0.5)$. |



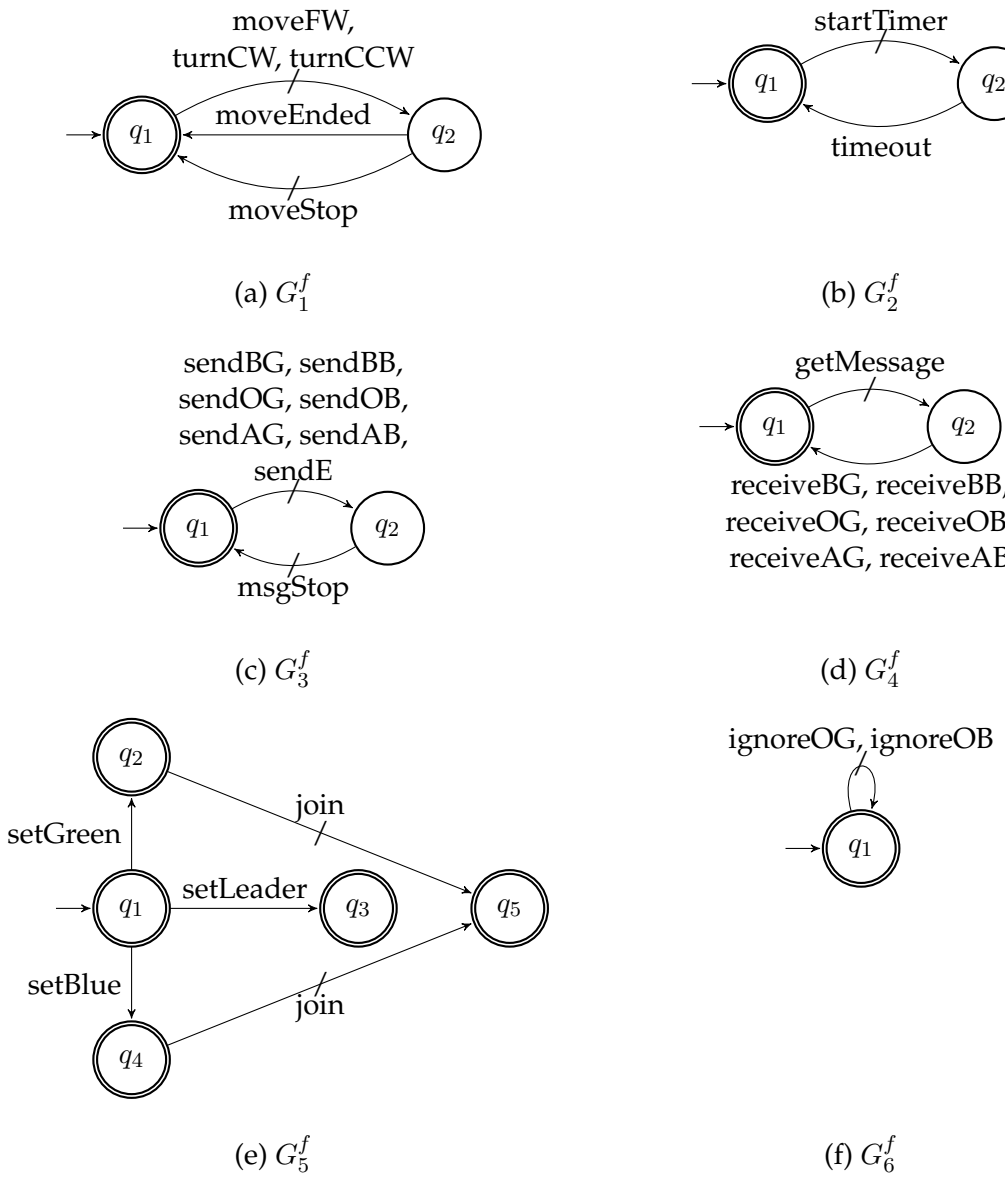

(a) $E_1^c$ (b) $E_2^c$ (c) $E_3^c$

(d) $E_4^c$ (e) $E_5^c$ (f) $E_6^c$

Figure 3.9: Specification for the object clustering case study. (a-c) Moves the robot according to the sensor reading; (d-f) prevents the same movement event from occurring consecutively (yet, the robot will perform its current movement indefinitely).

Figure 3.9 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).

### 3.1.5 Group formation




(a) $G_1^f$ (b) $G_2^f$

(c) $G_3^f$ (d) $G_4^f$

(e) $G_5^f$ (f) $G_6^f$

Figure 3.10: 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.

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 ($\pm 1$). 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 3.10 shows the free behaviour models for this strategy. In Table 3.5 the events' definition is summarised. Free behaviour model $G_1^f$ represents the movement capabilities. It is identical to $G_3^s$ in Figure 3.3(c).

Free behaviour model $G_2^f$ represents a timer. Controllable event *startTimer* starts the timer. After the defined time has elapsed, uncontrollable event *timeout* is generated.

Free behaviour model $G_3^f$ 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.

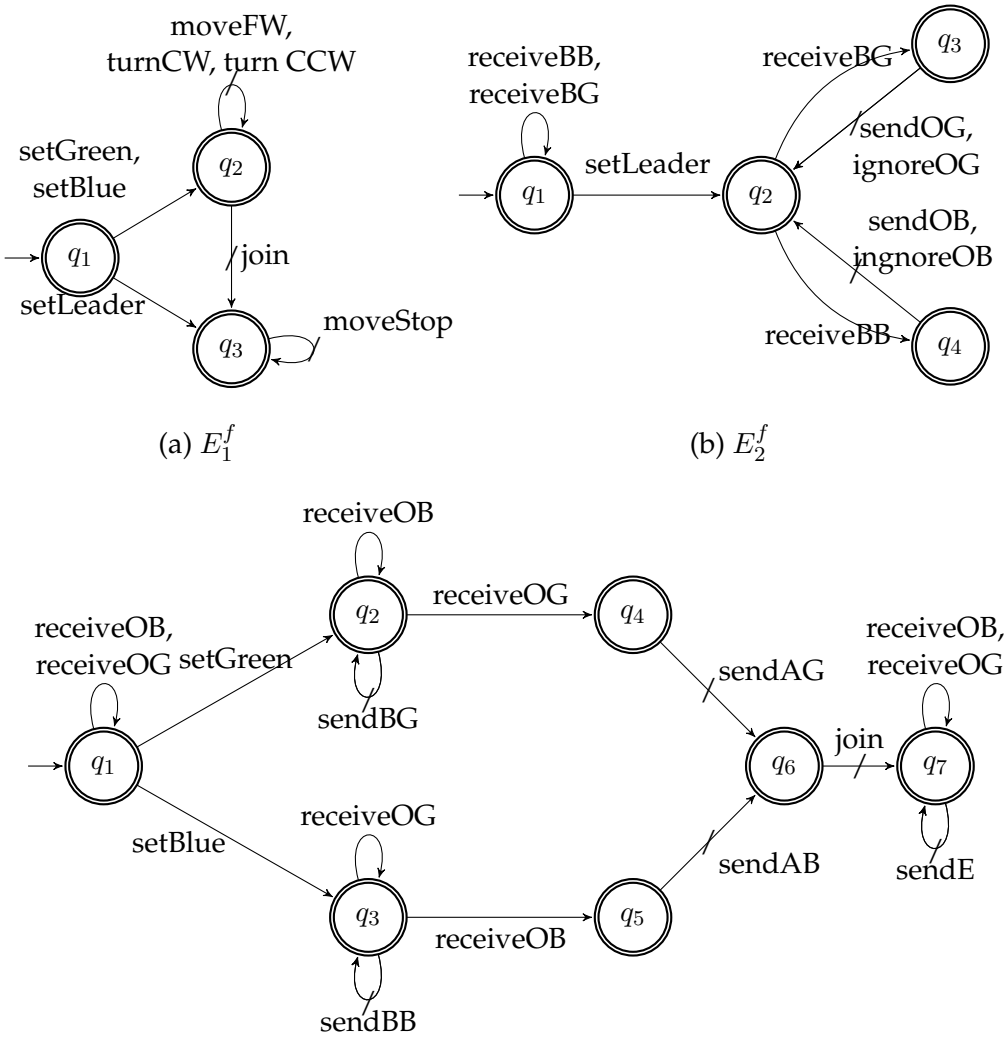
Free behaviour model $G_4^f$ defines the message receiving capability. Controllable event *getMessage* reads the most recent received message in the buffer. Uncontrollable events *receiveBG* and *receiveBB* are triggered when a broadcast message of a blue or green follower is received; *receiveOG* and *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 $G_5^f$ 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

Table 3.5: Summary of events' definition for the group formation strategy.

| event | controllability | definition |
|---|---|---|
| moveFW | *C* | Robot starts to move forward and continues doing so for a random period of time. |
| turnCW | *C* | Robot starts to turn clockwise and continues doing so for a random period of time. |
| turnCCW | *C* | Robot starts to turn counter-clockwise and continues doing so for a random period of time. |
| moveEnded | *U* | Signals that the last movement's period has ended and the robot stopped. |
| moveStop | *C* | Stops the robot (the controller forces the robot to stop ignoring any time left). |
| startTimer | *C* | Start the timer. |
| timeout | *U* | Signals the trigger of the timer. |
| sendBG | *C* | Broadcasts availability of green followers. |
| sendBB | *C* | Broadcasts availability of green followers. |
| sendOG | *C* | Offers membership to green follower. |
| sendOB | *C* | Offers membership to blue follower. |
| sendAG | *C* | Confirms the acceptance of a green follower. |
| sendAB | *C* | Confirms the acceptance of a blue follower. |
| sendE | *C* | Requests the relay of the last received message. |
| msgStop | *C* | stops sending current message. |
| getMessage | *C* | Reads the most recent received message. |
| receiveBG | *U* | Signals the reception of a broadcast message from a blue follower. |
| receiveBB | *U* | Signals the reception of a broadcast message from a green follower. |
| receiveOG | *U* | Signals the reception by a green follower of an offer to join the group. |
| receiveOB | *U* | Signals the reception by a blue follower of an offer to join the group. |
| receiveAG | *U* | Signals the reception of a acceptance message from a green follower. |
| receiveAB | *U* | Signals the reception of a acceptance message from a blue follower. |
| setGreen | *U* | Signals the configuration of the robot as a green follower. |
| setBlue | *U* | Signals the configuration of the robot as a blue follower. |
| setLeader | *U* | Signals the configuration of the robot as a leader. |
| join | *C* | Marks the robots a part of a group. |
| ignoreOG | *C* | Leader do not send an offer, ignoring a green broadcast. |
| ignoreOB | *C* | Leader do not send an offer, ignoring a blue broadcast. |

randomly selected during initialisation. Followers can join a group triggering the controllable event *join*. In the free behaviour model $G_6^f$, the leader can choose not to send an offer by the controllable events *ignoreOG* and *ignoreOB*. This is used to implement the equilibrium criterion.




(a) $E_1^f$ (b) $E_2^f$

(c) $E_3^f$

Figure 3.11: 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.

Figures 3.11 and 3.12 illustrates the specifications for the group formation strategy. Specification $E_1^f$ defines that followers that have not yet joined any group can move. All other robots are not allowed to move. Specification $E_2^f$ defines that a leader robot can send an offer to a follower when it receives a corresponding broadcast message. Al-

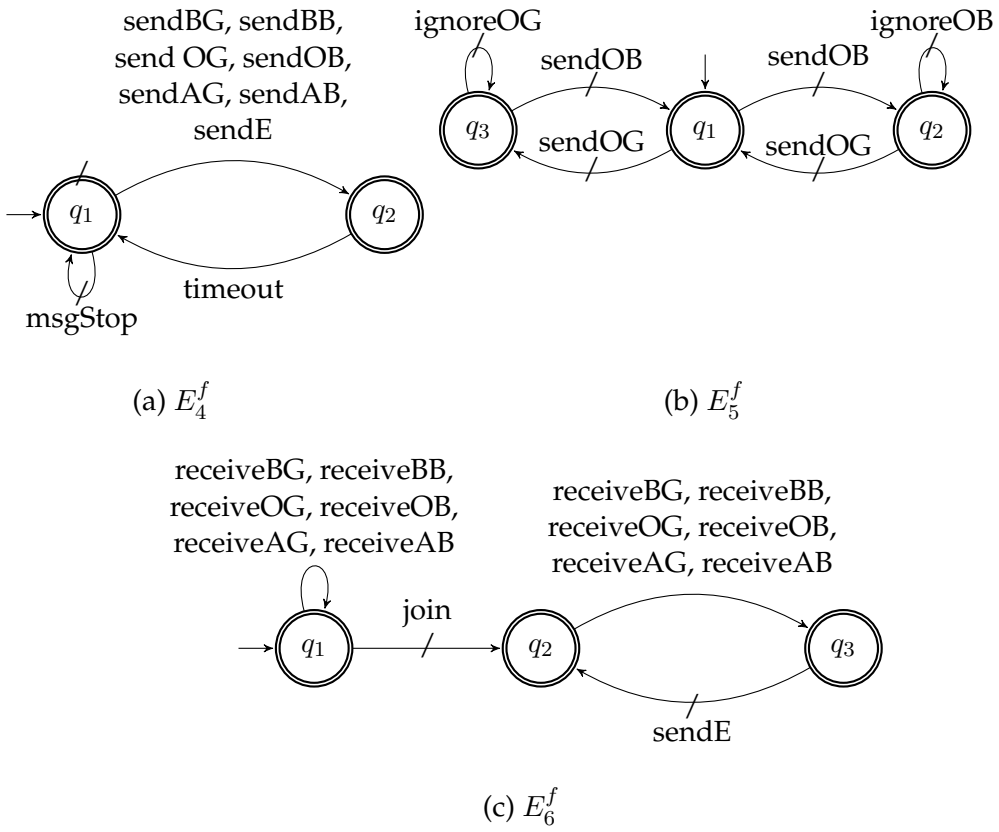



Figure 3.12: Specification models for the group formation case study. (a) Ensures that a message is transmitted for a minimum period; (b) guarantees the equilibrium criterion; (c) the robot's ability to choose not to make an offer.

ternatively, it can choose to ignore the request. Specification $E_3^f$ 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 $E_4^f$ controls the transmission mechanisms. A message is transmitted by the subsystem over a predefined period of time. Once a message is sent, it can only be stopped after the timeout of the subsystem. Specification $E_5^f$ 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 $E_6^f$ 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 Section 4.1); this is not implemented by using the formal method.

### 3.1.6 Design guidelines

In this section, we provide some guidelines for the design of the system's free behaviour models and control specifications.

- Initial states are often the only marked state in free behaviour models, as it implies that the system must return to its initial condition. Exceptions to this rules exist, for example in model $G_5^f$ of the group formation (Figure 3.10(e)) where transitions represent a change in configuration/role that will persist indefinitely.
- In control specifications, all states are usually marked.
- Use single-state free behaviour models with uncontrollable self-loops when a sensor can give an input at any time and with controllable self-loops when the controller can trigger an action at any time.
- State machines use to represent ad-hoc implementations usually have actions (often related to controllable events) triggered by states while transitions only occur based on sensing (often related to uncontrollable events). With generators, a similar approach can be achieved by replacing the actions in each state with self-loop triggered by the controllable event.
- Generators make possible the use of specifications that have a transition with a controllable event from one state to another (instead of a self-loop) this allows the designer to separate the state space not only regarding observed input (uncontrollable events) but control responses (controllable events).
- Design the specifications regarding what must **not** occur in a particular state.
- Not only controllable but also uncontrollable events can be disabled in a particular state, the supervisor synthesis (in the next section) will then calculate which

controllable events must be disabled in which state to guarantee that undesired uncontrollable events do not occur. When disabling uncontrollable events, you may obtain an empty controller (a supervisor with no states and transitions) as any control response of the controller will achieve a state where the occurrence of an undesired event is possible.

- Each free behaviour model represents a small subsystem of your system (the robot), and each control specification describes a particular aspect relating two or more subsystem.

## 3.2 Supervisor synthesis

Now the synthesis of supervisors for the case studies are presented.

### 3.2.1 Monolithic

The first step to synthesise a monolithic supervisor is to compose all free behaviour models in a unique generator. The composition for all cases are shown as follows: Equation 3.2 for the orbit; Equation 3.3 for the segregation; Equation 3.4 for the aggregation; Equation 3.5 for the object clustering; and Equation 3.6 for the group formation.

$$G^o = G^o_1 || \cdots || G^o_4 \tag{3.2}$$

$$G^s = G^s_1 || \cdots || G^s_4 \tag{3.3}$$

$$G^a = G^a_1 || G^a_2 \tag{3.4}$$

$$G^c = G^c_1 || G^c_2 \tag{3.5}$$

$$G^f = G^f_1 || \cdots || G^f_6 \tag{3.6}$$

The specifications are also composed in a unique generator. The composition for all cases are shown as follows: Equation 3.7 for the orbit; Equation 3.8 for the segregation; Equation 3.9 for the aggregation; Equation 3.10 for the object clustering; and Equation 3.11 for the group formation.

$$E^o = E^o_1 || \cdots || E^o_3 \tag{3.7}$$

$$E^s = E^s_1 || \cdots || E^s_3 \tag{3.8}$$

$$E^a = E^a_1 || \cdots || E^a_4 \tag{3.9}$$

$$E^c = E^c_1 || \cdots || E^c_6 \tag{3.10}$$

$$E^f = E^f_1 || \cdots || E^f_6 \tag{3.11}$$

The monolithic target language, $K^\theta$, is obtained by the synchronous composition of $G^\theta$ and $E^\theta$:[2]

$$K^\theta = G^\theta || E^\theta. \tag{3.12}$$

Finally, the monolithic supervisor $S^\theta$ is obtained as:

$$S^\theta \ : \ L_m(S^\theta / G^\theta) = SupC(G^\theta, K^\theta). \tag{3.13}$$

### 3.2.2 Modular

Rather than calculating a single target language, a target language $K^{mod,\theta}_j$ is calculated for each specification $E^\theta_j$. The free behaviour model used, $G^{mod,\theta}$, is the same of the

[2]Note that $\theta$ denotes the case study as defined in Section 3.1.

monolithic approach (i.e. $G^{mod,\theta} = G^{\theta}$). Equation 3.14 shows the calculation of the target language for the orbit; Equation 3.15 for the segregation; Equation 3.16 for the aggregation; Equation 3.17 for the object clustering; and Equation 3.18 for the group formation.

$$K_j^{mod,o} = G^{mod,o} || E_j^o \quad \forall\, j \in \{1,2,3\} \tag{3.14}$$

$$K_j^{mod,s} = G^{mod,s} || E_j^s \quad \forall\, j \in \{1,2,3\} \tag{3.15}$$

$$K_j^{mod,a} = G^{mod,a} || E_j^a \quad \forall\, j \in \{1,\cdots,4\} \tag{3.16}$$

$$K_j^{mod,c} = G^{mod,c} || E_j^c \quad \forall\, j \in \{1,\cdots,6\} \tag{3.17}$$

$$K_j^{mod,f} = G^{mod,f} || E_j^f \quad \forall\, j \in \{1,\cdots,6\} \tag{3.18}$$

One modular supervisor is obtained for each target language. The obtained supervisor is the one that generates the maximal controllable languages of $K_j^{mod,\theta}$ over $G^{mod,\theta}$. Equation 3.19 for the orbit; Equation 3.20 for the segregation; Equation 3.21 for the aggregation; Equation 3.22 for the object clustering; and Equation 3.23 for the group formation.

$$S_j^{mod,o} : L_m(S_j^{mod,o}/G^{mod,o}) = SupC(K_j^{mod,o}, G^{mod,o}) \quad \forall\, j \in \{1,2,3\} \tag{3.19}$$

$$S_j^{mod,s} : L_m(S_j^{mod,s}/G^{mod,s}) = SupC(K_j^{mod,s}, G^{mod,s}) \quad \forall\, j \in \{1,2,3\} \tag{3.20}$$

$$S_j^{mod,a} : L_m(S_j^{mod,a}/G^{mod,c}) = SupC(K_j^{mod,a}, G^{mod,c}) \quad \forall\, j \in \{1,\cdots,4\} \tag{3.21}$$

$$S_j^{mod,c} : L_m(S_j^{mod,c}/G^{mod,c}) = SupC(K_j^{mod,c}, G^{mod,c}) \quad \forall\, j \in \{1,\cdots,6\} \tag{3.22}$$

$$S_j^{mod,f} : L_m(S_j^{mod,f}/G^{mod,f}) = SupC(K_j^{mod,f}, G^{mod,f}) \quad \forall\, j \in \{1, \cdots, 6\} \tag{3.23}$$

### 3.2.3 Local modular

In the local modular approach, similar to the modular one, a supervisor $S_j^{loc,\theta}$ is created for each control specification $E_j^{loc,\theta}$ using a local free behaviour model $G_j^{loc,\theta}$.

#### 3.2.3.1 Orbit

Table 3.6: Events used by the specifications and free behaviour models for the orbit case study. 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.

| | | $E_1^o$ | $E_2^o$ | $E_3^o$ |
|---|---|---|---|---|
| $G_1^o$ | *press* | ✓ | | |
| $G_2^o$ | *static* | ✓ | ✓ | |
| | *orbit* | ✓ | ✓ | |
| $G_3^o$ | *turnCW* | | ✓ | ✓ |
| | *turnCCW* | | ✓ | ✓ |
| | *moveFW* | | ✓ | ✓ |
| $G_4^o$ | *upperbound* | | | ✓ |
| | *bound* | | | ✓ |
| | *lowerbound* | | | ✓ |
| local models | | $G_1^{loc,o}$ | $G_2^{loc,o}$ | $G_3^{loc,o}$ |

Table 3.6 shows the relation of events for each specification for the orbit case study. The local free behaviour models are obtained as:

$$\begin{aligned} G_1^{loc,o} &= G_1^o || G_2^o, \\ G_2^{loc,o} &= G_2^o || G_3^o, \\ G_3^{loc,o} &= G_3^o || G_4^o. \end{aligned} \tag{3.24}$$

The target languages are:

$$\begin{aligned} K_1^{loc,o} &= G_1^{loc,o} || E_1^{loc,o}, \\ K_2^{loc,o} &= G_2^{loc,o} || E_2^{loc,o}, \\ K_3^{loc,o} &= G_3^{loc,o} || E_3^{loc,o}. \end{aligned} \tag{3.25}$$

The local modular supervisors are:

$$\begin{aligned} S_1^{loc,o} &= SupC(G_1^{loc,o} || K_1^{loc,o}), \\ S_2^{loc,o} &= SupC(G_2^{loc,o} || K_2^{loc,o}), \\ S_3^{loc,o} &= SupC(G_3^{loc,o} || K_3^{loc,o}). \end{aligned} \tag{3.26}$$

#### 3.2.3.2 Segregation

Table 3.7: Events used by the specifications and free behaviour models for the segregation case study.

| | | $E_1^s$ | $E_2^s$ | $E_3^s$ |
|---|---|---|---|---|
| $G_1^s$ | *press* | ✓ | | |
| $G_2^s$ | *sendR* | ✓ | ✓ | |
| | *sendG* | ✓ | ✓ | |
| | *sendB* | ✓ | ✓ | |
| | *sendNothing* | ✓ | ✓ | |
| $G_3^s$ | *moveFW* | | ✓ | ✓ |
| | *turnCW* | | ✓ | ✓ |
| | *turnCCW* | | ✓ | ✓ |
| | *moveEnded* | | | |
| | *moveStop* | | | ✓ |
| $G_4^s$ | *getR* | | | ✓ |
| | *getNotR* | | | ✓ |
| $G_5^s$ | *getG* | | | ✓ |
| | *getNotG* | | | ✓ |
| $G_6^s$ | *getB* | | | ✓ |
| | *getNotB* | | | ✓ |
| local models | | $G_1^{loc,s}$ | $G_2^{loc,s}$ | $G_3^{loc,s}$ |

Table 3.7 shows the relation of events for each specification for the segregation case

study. The local free behaviour models are obtained as:

$$\begin{array}{lcl} G_1^{loc,s} & = & G_1^s||G_2^s, \\ G_2^{loc,s} & = & G_2^s||G_3^s, \\ G_3^{loc,s} & = & G_3^s||\ldots||G_6^s. \end{array} \tag{3.27}$$

The target languages are:

$$\begin{array}{lcl} K_1^{loc,s} & = & G_1^{loc,s}||E_1^{loc,s}, \\ K_2^{loc,s} & = & G_2^{loc,s}||E_2^{loc,s}, \\ K_3^{loc,s} & = & G_3^{loc,s}||E_3^{loc,s}. \end{array} \tag{3.28}$$

The local modular supervisors are:

$$\begin{array}{lcl} S_1^{loc,s} & = & SupC(G_1^{loc,s}||K_1^{loc,s}), \\ S_2^{loc,s} & = & SupC(G_2^{loc,s}||K_2^{loc,s}), \\ S_3^{loc,s} & = & SupC(G_3^{loc,s}||K_3^{loc,s}). \end{array} \tag{3.29}$$

#### 3.2.3.3 Aggregation

Table 3.8: 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 3.8 shows the relation of events for each specification for the aggregation case study. The local free behaviour models are obtained as:

$$\begin{array}{lcl} G_1^{loc,a} = G_2^{loc,a} & = & G_1^a||G_2^a, \\ G_3^{loc,a} = G_4^{loc,a} & = & G_2^a. \end{array} \tag{3.30}$$

The target languages are:

$$\begin{array}{rcl} K_1^{loc,a} & = & G_1^{loc,a} || E_1^{loc,a}, \\ K_2^{loc,a} & = & G_2^{loc,a} || E_2^{loc,a}, \\ K_3^{loc,a} & = & G_3^{loc,a} || E_3^{loc,a}, \\ K_4^{loc,a} & = & G_4^{loc,a} || E_4^{loc,a}. \end{array} \tag{3.31}$$

The local modular supervisors are:

$$\begin{array}{rcl} S_1^{loc,a} & = & SupC(G_1^{loc,a} || K_1^{loc,a}), \\ S_2^{loc,a} & = & SupC(G_2^{loc,a} || K_2^{loc,a}), \\ S_3^{loc,a} & = & SupC(G_3^{loc,a} || K_3^{loc,a}), \\ S_4^{loc,a} & = & SupC(G_4^{loc,a} || K_4^{loc,a}). \end{array} \tag{3.32}$$

#### 3.2.3.4 Object clustering

Table 3.9: 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 | | $G_1^{loc,c}$ | $G_2^{loc,c}$ | $G_3^{loc,c}$ | $G_4^{loc,c}$ | $G_5^{loc,c}$ | $G_6^{loc,c}$ |

Table 3.9 shows the relation of events for each specification for the object clustering case study. The local free behaviour models are obtained as:

$$\begin{array}{rcl} G_1^{loc,c} = G_2^{loc,c} = G_3^{loc,c} & = & G_1^c || G_2^c, \\ G_4^{loc,c} = G_5^{loc,c} = G_6^{loc,c} & = & G_2^c. \end{array} \tag{3.33}$$

The target languages are:

$$\begin{array}{lcl} K_1^{loc,c} & = & G_1^{loc,c}||E_1^{loc,c}, \\ K_2^{loc,c} & = & G_2^{loc,c}||E_2^{loc,c}, \\ K_3^{loc,c} & = & G_3^{loc,c}||E_3^{loc,c}, \\ K_4^{loc,c} & = & G_4^{loc,c}||E_4^{loc,c}, \\ K_5^{loc,c} & = & G_5^{loc,c}||E_5^{loc,c}, \\ K_6^{loc,c} & = & G_6^{loc,c}||E_6^{loc,c}. \end{array} \tag{3.34}$$

The local modular supervisors are:

$$\begin{array}{lcl} S_1^{loc,c} & = & SupC(G_1^{loc,c}||K_1^{loc,c}), \\ S_2^{loc,c} & = & SupC(G_2^{loc,c}||K_2^{loc,c}), \\ S_3^{loc,c} & = & SupC(G_3^{loc,c}||K_3^{loc,c}), \\ S_4^{loc,c} & = & SupC(G_4^{loc,c}||K_4^{loc,c}), \\ S_5^{loc,c} & = & SupC(G_5^{loc,c}||K_5^{loc,c}), \\ S_6^{loc,c} & = & SupC(G_6^{loc,c}||K_6^{loc,c}). \end{array} \tag{3.35}$$

#### 3.2.3.5 Group formation

Table 3.10 shows the relation of events for each specification for the group formation case study. The local free behaviour models are obtained as:

$$\begin{array}{lcl} G_1^{loc,f} & = & G_1^f||G_5^f, \\ G_2^{loc,f} & = & G_3^f||\cdots||G_6^f, \\ G_3^{loc,f} = G_6^{loc,f} & = & G_3^f||G_4^f||G_5^f, \\ G_4^{loc,f} & = & G_2^f||G_3^f, \\ G_5^{loc,f} & = & G_3^f||G_6^f. \end{array} \tag{3.36}$$

Table 3.10: Events used by specifications and free behaviour models for the group formation case study.

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

The target languages are:

$$\begin{aligned} K_1^{loc,f} &= G_1^{loc,f} || E_1^{loc,f}, \\ K_2^{loc,f} &= G_2^{loc,f} || E_2^{loc,f}, \\ K_3^{loc,f} &= G_3^{loc,f} || E_3^{loc,f}, \\ K_4^{loc,f} &= G_4^{loc,f} || E_4^{loc,f}, \\ K_5^{loc,f} &= G_5^{loc,f} || E_5^{loc,f}, \\ K_6^{loc,f} &= G_6^{loc,f} || E_6^{loc,f}. \end{aligned} \tag{3.37}$$

The local modular supervisors are:

$$\begin{aligned} S_1^{loc,f} &= SupC(G_1^{loc,f}||K_1^{loc,f}), \\ S_2^{loc,f} &= SupC(G_2^{loc,f}||K_2^{loc,f}), \\ S_3^{loc,f} &= SupC(G_3^{loc,f}||K_3^{loc,f}), \\ S_4^{loc,f} &= SupC(G_4^{loc,f}||K_4^{loc,f}), \\ S_5^{loc,f} &= SupC(G_5^{loc,f}||K_5^{loc,f}), \\ S_6^{loc,f} &= SupC(G_6^{loc,f}||K_6^{loc,f}). \end{aligned} \tag{3.38}$$

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

$$K_j^{loc,\theta} = G_j^{loc,\theta}||E_j^{\theta} \quad \forall\, j \in \{1, \ldots, n^{\theta}\}. \tag{3.39}$$

The local modular supervisor $S_j^{loc,\theta}$ is obtained for each target language $K_j^{loc,\theta}$, analogous to Equation 2.30:

$$S_j^{loc,\theta} : L_m(S_j^{loc,\theta}/G_j^{loc,\theta}) = SupC(G_j^{loc,\theta}, K_j^{loc,\theta}) \quad \forall\, j \in \{1, \ldots, n^{\theta}\}. \tag{3.40}$$

#### 3.2.3.6 Enabled events

In the case of multiple supervisors $S_1, \ldots, S_n$ 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.

### 3.2.4 Comparison

Table 3.11 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 [135]). The local modular approach turned out to be the most memory efficient in three of the four case studies. In the remaining

Table 3.11: Total number of states and transitions for each case study when using monolithic, modular, and local modular synthesis approaches, respectively. Data corresponds 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.

| | | Monolithic | | Modular | | Local modular | |
|---|---|---|---|---|---|---|---|
| | | K | S | K | S | K | S |
| Orbit | states | 12 | 12 | 12 | 12 | **9** | **9** |
| | transitions | 60 | 60 | 79 | 79 | **23** | **23** |
| 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 (304) | 130 (138) | 560 (776) | 488 (680) | **93** (108) | **79** (92) |
| | transitions | 1446 (1602) | 531 (561) | 6708 (9272) | 5768 (8012) | **553** (641) | **452** (525) |

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 3.11, the local modular approach often outperforms the other approaches in terms of total number of states and transitions (and hence in memory usage).

## 3.3 Summary

This chapter demonstrated the use of supervisory control theory (SCT) for formally developing controllers of swarm robotics systems. 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—could then be derived from these models. The supervisors, here represented as regular languages, were 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 computational tools [136, 137, 138, 139].

# 4

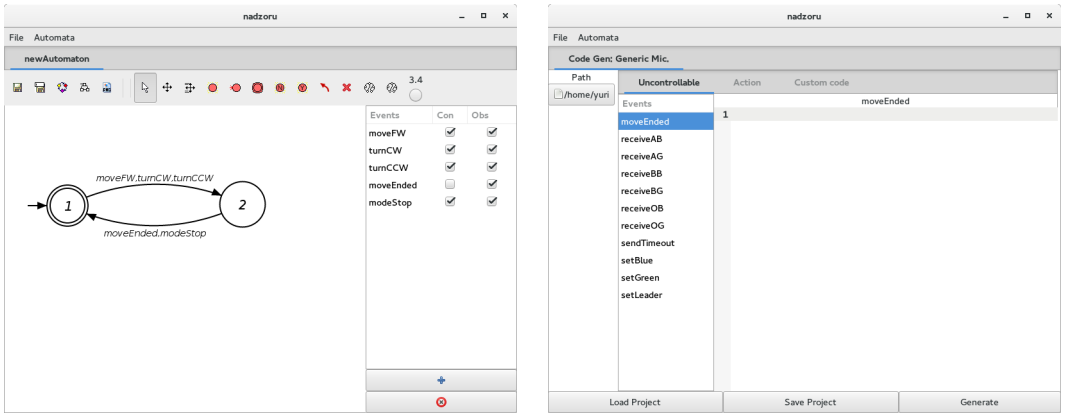
# A framework for executing supervisors on swarms of robots

In this chapter the implementation of formal controllers is discussed. First, the representation of the supervisors in the memory is described. Second, it is presented how these supervisors can be executed by a generator player, taking in account the occurrences of uncontrollable events. Third, it is shown how the controllable events of the supervisor can be translated to hardware actions and how the reading of the sensors can be translated to uncontrollable events—this is performed by what is called the operational procedures. Finally, experiments for all the case studies from the previous chapter are performed using the proposed implementation.

## 4.1 Implementation of supervisory control in swarm robotics

The implementation of swarm robotics' control strategies is, in general, challenging as errors are commonly introduced at this step. This is particularly the case if the strategies are not formally defined and are implemented in an ad-hoc manner. The exercise of formalising the strategy before implementation can assist in the identification of potential flaws. For example, in the implementation of the group formation strategy (see Section 3.1.5) one may not realise that the leader may have to ignore certain broadcast messages it receives; or that the followers that joined a group must also stop any current movement in addition to not triggering any new movements. To further improve the translation of formally defined controllers to source code, this section presents a framework for the automatic code generation for swarm robotics' controllers.

The implementation is based on the SCT architecture proposed by [20, 22]. It adds three layers on top of the robot's hardware: the supervisor (which is at the highest level), the




(a) (b)

Figure 4.1: Nadzoru is an open source software tool for SCT [1, 140, 27]. It can be used for all stages, from modelling (a) to automatic code generation (b).

generator player, and then the operational procedures. In this thesis, we implement complete local modular supervisors.[1]

The implementation uses the open source software tool Nadzoru [1, 140, 27] 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.

Figure 4.1 shows two snapshots of the Nadzoru graphical user interface. A video demonstrating its use is available in the electronic supplementary material.

### 4.1.1 Supervisor representation in memory

Figure 4.2 illustrates the data structure that stores the synthesised supervisor in memory [1]. Each state is represented as a block of this data structure. Each block describes all output transitions from that state. The first byte for 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

[1] Note that these differ from reduced local modular supervisors [20, 22].

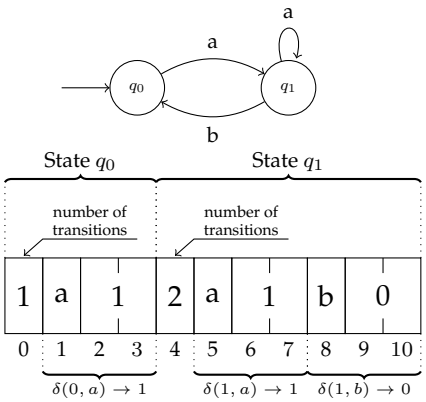



Figure 4.2: Supervisor data structure in memory [1]. The first element of each state represents the number of outgoing transitions. It is followed by blocks of three elements, which detail the event that triggers the transition and the resulting state.

output transitions per state. The memory occupation (in bytes) of the supervisor data structure is given by:

$$mem = s + 3 \times t, \tag{4.1}$$

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 [1].

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 block 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.

### 4.1.2 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.

**Algorithm 1** Generator player

```
1: Let N be the number of generators
2: procedure GENERATOR PLAYER
3:     for all j ∈ {1, 2, ..., N} do
4:         set current state c_j to initial state q_{0_j};
5:     end for
6:     while true do
7:         if uncontrollable event e_u ∈ Σ_u occurred then
8:             for all j ∈ {1, 2, ..., N} do
9:                 c_j = δ_j(c_j, e_u);
10:            end for
11:        else
12:            calculate the set of enabled controllable events ψ(c_1, c_2, ..., c_n);
13:            if ψ(c_1, c_2, ..., c_n) ≠ ∅ then
14:                select one controllable event e_c ∈ ψ(c_1, c_2, ..., c_n);
15:                for all j ∈ {1, 2, ..., N} do
16:                    c_j = δ_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
```

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 Section 4.1.3). 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.

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.

### 4.1.3 Operational procedures

The operational procedures are a low-level interface of the controller to the hardware [22]. 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() % 5);
    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;
}
```

In the following the operational procedure implementation of the case studies is presented.

Table 4.1: Memory usage of the control software for each case study and robot platform using the local modular approach. Values are in bytes.

| Case | Platform | Libraries/ Base code | Operational procedures | Generator player | Super-visors |
|---|---|---|---|---|---|
| Orbit | e-puck | 25344 | 948 | 2112 | 216 |
| Orbit | Kilobot | 10968 | 1266 | 1482 | 126 |
| Segregation | e-puck | 25308 | 1773 | 2106 | 654 |
| Segregation | Kilobot | 10968 | 1512 | 1478 | 418 |
| Aggregation | e-puck | 17502 | 408 | 2016 | 207 |
| Object clustering | e-puck | 16695 | 696 | 2004 | 378 |
| Group formation | Kilobot | 10968 | 1742 | 1584 | 1704 |

### 4.1.4 Memory usage

Table 4.1 shows the memory usage of the control software using the proposed implementation for each case study and robot platform using the local modular approach. 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 Section 4.1.1; the values depend on the specific overhead and implementation details of each compiler.

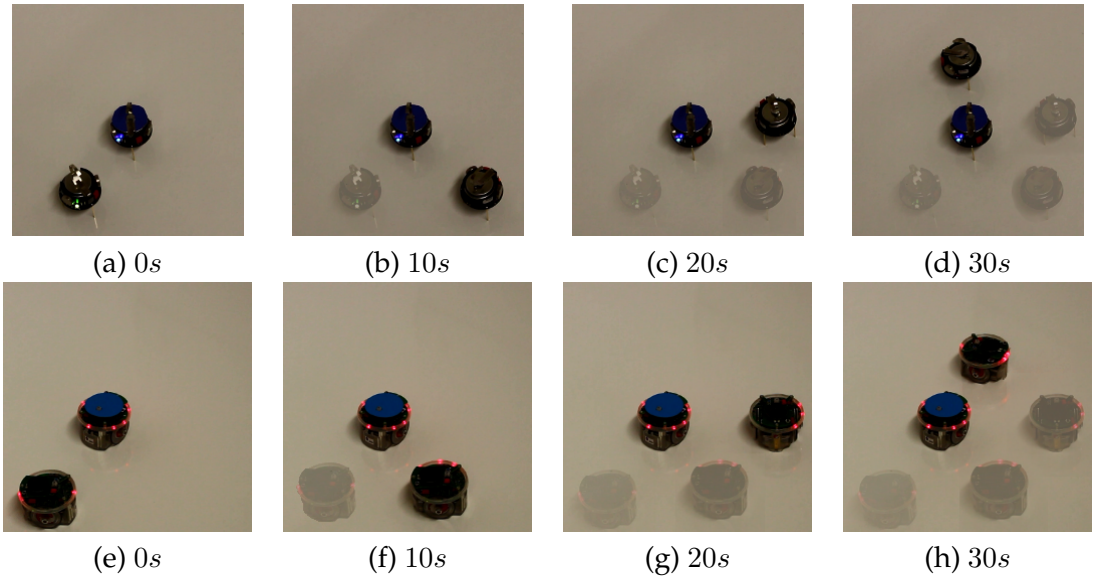
## 4.2 Experiments

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 on the online supplementary material [141].

### 4.2.1 Orbit

The experiment took place in a two-dimensional $1.20\ m \times 0.90\ m$ arena. It used two robots, a static robot is placed in the centre of the arena and an orbiting robot is placed inside the configured boundaries. Ten trials were performed for each platform. Each trial was limited to $300\ s$.

Visual inspection confirmed that the SCT controller performed the orbit task as intended. Figure 4.3 shows snapshots taken every $10\ s$ for the first $30\ s$ from one of the experimental trials with the Kilobots and e-pucks, respectively.



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

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

Figure 4.3: 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.

### 4.2.2 Segregation

The segregation experiment took place in a two-dimensional $1.20$ m $\times$ $0.90$ 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 4.4 shows snapshots taken from one of the experimental trials with the Kilobots and e-pucks, respectively. Figures 4.4(a) and (c) show the initial grid formation with leaders marked using tags. Figures 4.4(b) and (d) show the result after segregation occurred. For Figure 4.4(d), tags were added to all robots after the experiments for visualisation purposes (based on the robots' states as indicated by their LEDs).

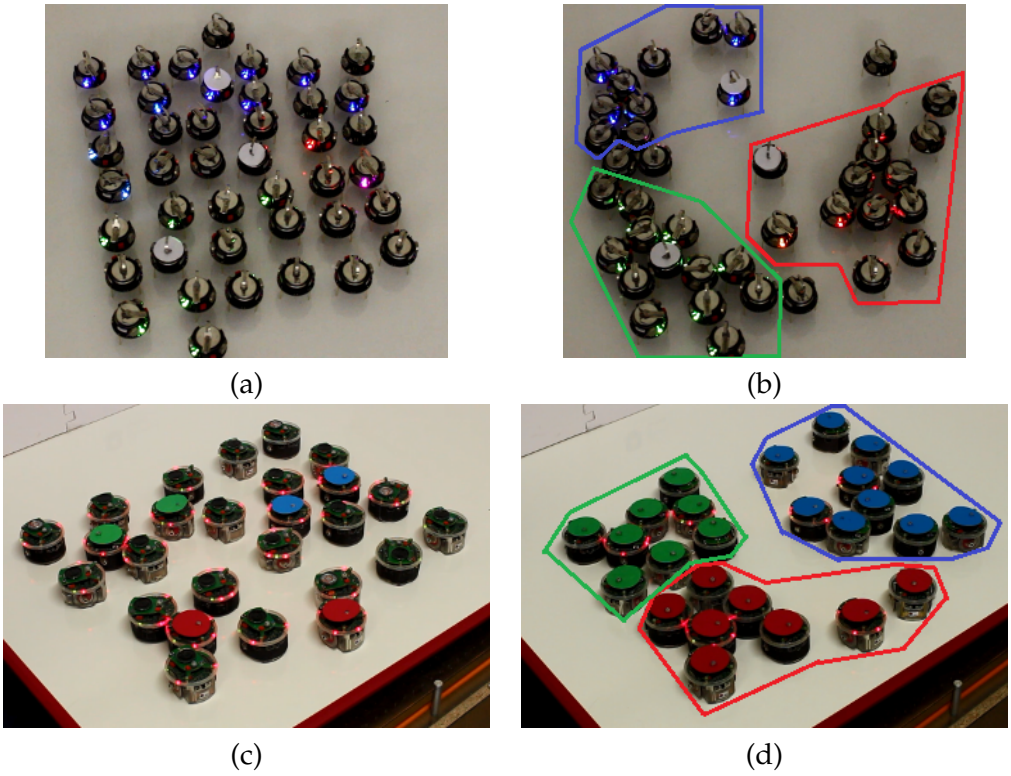


Figure 4.4: Snapshots from a segregation trial with Kilobots: (a) initial grid formation with three leaders, marked with white tags; (b) result after segregation occurred. Trial with e-pucks: (c) initial grid formation with three pairs of leaders marked with tags; (d) result after segregation occurred, tags were added after the experiments for visualisation.

### 4.2.3 Aggregation

The aggregation experiment used the same configuration as [53]. 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 $640 \times 480$ and a field of view of $56° \times 42°$. To simplify identification, the robots were fitted with black skirts. Trials were performed in a $400$ cm $\times$ $225$ cm light grey floor arena surrounded by white walls that were $50$ cm in height. The arena had $120$ pencil marks distributed as a $15 \times 8$ 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 [53]). This provided reliable readings in a range of up to $150\ cm$ [53].

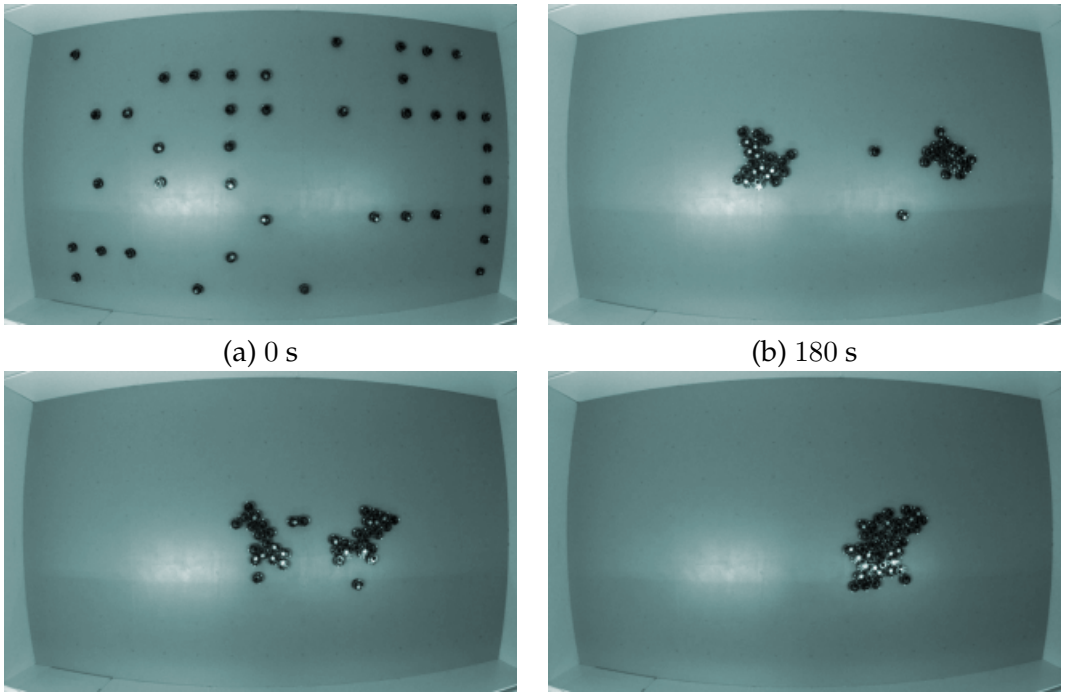
Ten trials were performed, each lasting $900$ s. An IR signal was emitted to instruct the robots to turn in place for a random period of time. As a result, the robots were randomly orientated. A second IR signal instructed the robots to start the controller. In case of failure of any robot (e.g. the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

Figure 4.5 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.

### 4.2.4 Object clustering

The object clustering experiment used the same configuration as in [49]. 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 [49]. 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 [49], 5 e-pucks and 20 objects were used. Figure 4.6 shows snapshots taken from one of the experimental trials.



(a) 0 s (b) 180 s

(c) 300 s (d) 480 s

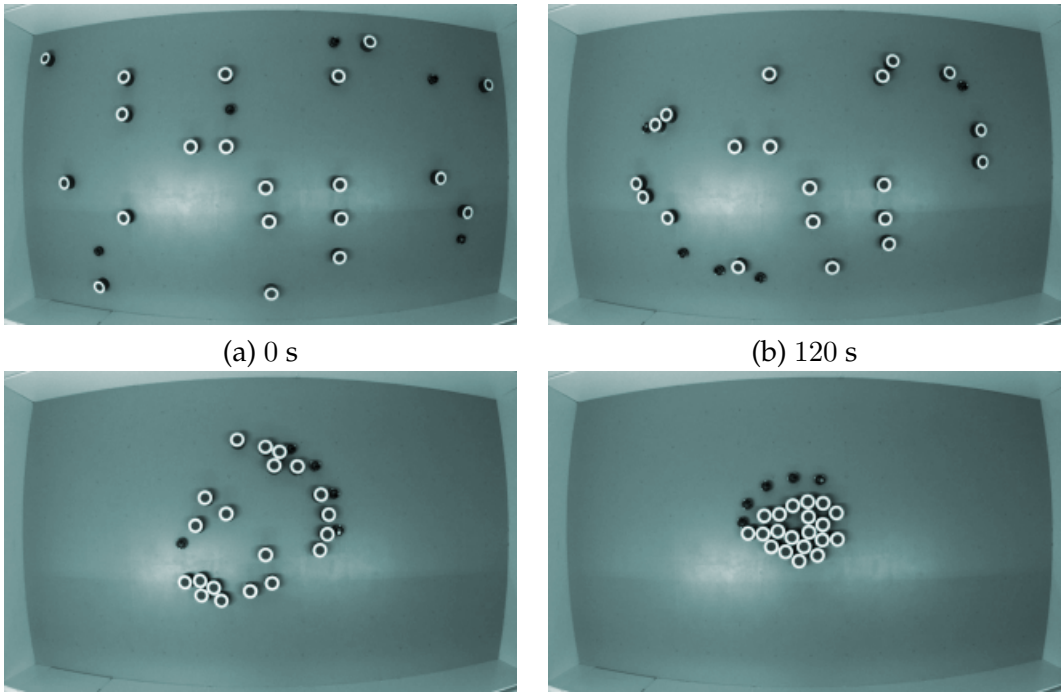
Figure 4.5: Sequence of snapshots taken from one of the trials where 40 e-pucks perform the aggregation task. Photo (a) displays the initial positions of the robots. Photos (b-d) show the experiment after 120 s, 300 s, and 480 s.

To evaluate whether the performance of object clustering using SCT is similar to the original implementation [49], 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 [49]:

$$u^{(t)} = \frac{1}{4r_o^2} \sum_{i=1}^{N_o} ||\mathbf{p}_i^{(t)} - \bar{\mathbf{p}}^{(t)}||^2, \tag{4.2}$$

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 Figure 4.7. The coloured curves correspond to the 10 trials, the black dashed line represents the mean. Figures 4.7(a) and (b) show the proportion of objects in the largest cluster. The results from [49] are plotted in (a), and



(a) 0 s (b) 120 s

(c) 300 s (d) 480 s

Figure 4.6: Sequence of snapshots taken from one of the trials in which 5 e-pucks—in black—cluster 20 objects—in white. Photo (a) displays the initial positions of the robots and the objects. Photos (b-d) show the experiment after 120 s, 300 s, and 480 s.

those obtained using SCT in (b). Figures 4.7(c) and (d) show the compactness of objects. The results from [49] are plotted in (c), and those obtained using SCT in (d). Overall the 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.

In Figure 4.8 we highlight the mean dynamics of all 10 trials—as previously detailed in Figure 4.7—and we indicate the standard deviation. Figures 4.8(a-b) displays the proportion of clustered objects over time using data from [49] and data collected using the controller that was synthesised using SCT, respectively. Figures 4.8(c-d) displays the compactness of objects over time using data from [49] and data collected using the controller that was synthesised using SCT, respectively.

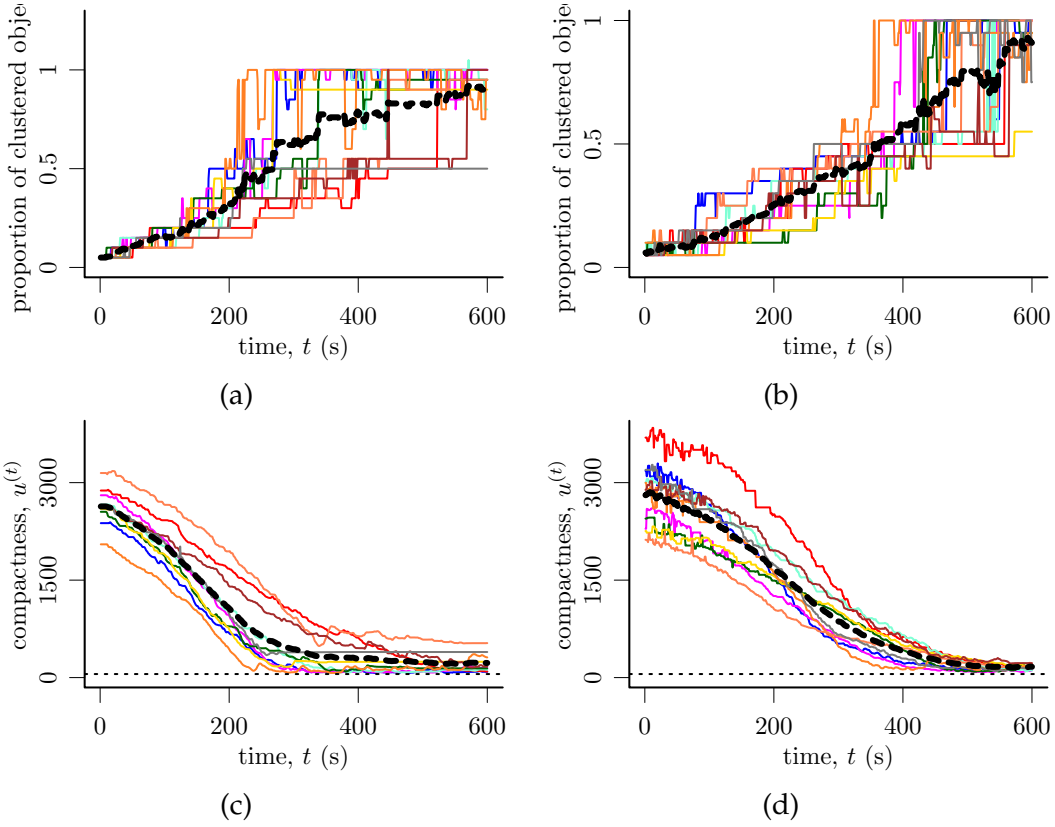



Figure 4.7: Dynamics of object clustering with 5 e-pucks and 20 objects. (a) and (c) are plotted using data from [49]. (b) and (d) present the results for the controller that was synthesised using SCT. In (a) and (b), the relationship between the proportion of clustered objects and time is displayed. (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 [49].

### 4.2.5 Group formation

The experiments took place in a $1.20\ m \times 0.90\ m$ two-dimensional white floor arena. The arena had $88$ marks distributed as a $11 \times 8$ grid, where columns and rows were $10\ cm$ away from each other. $88$ Kilobots were placed on each pencil marks. Ten trials were performed and each trial was limited to $300\ s$. The experiment evaluates the match of the modelled specifications with the synthesised control logic. We observed that the robots behaved according to the specifications in all the trials. Figure 4.9 shows snapshots taken from the experimental trials with the Kilobots.

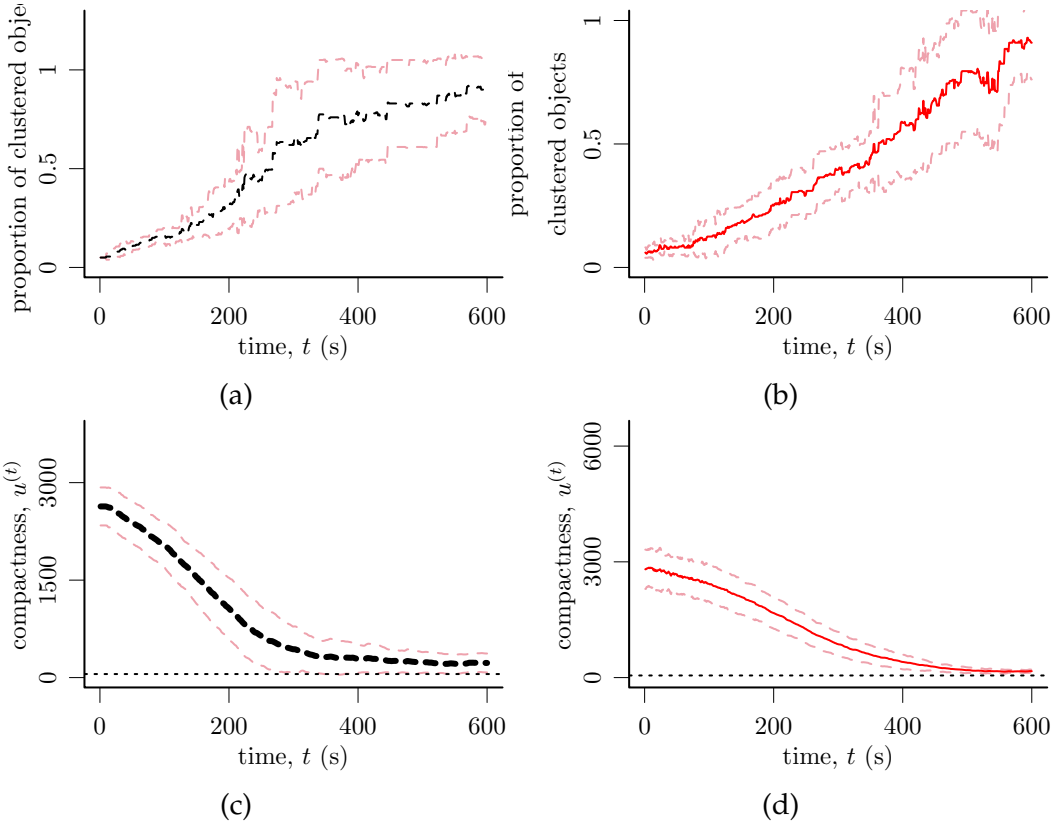
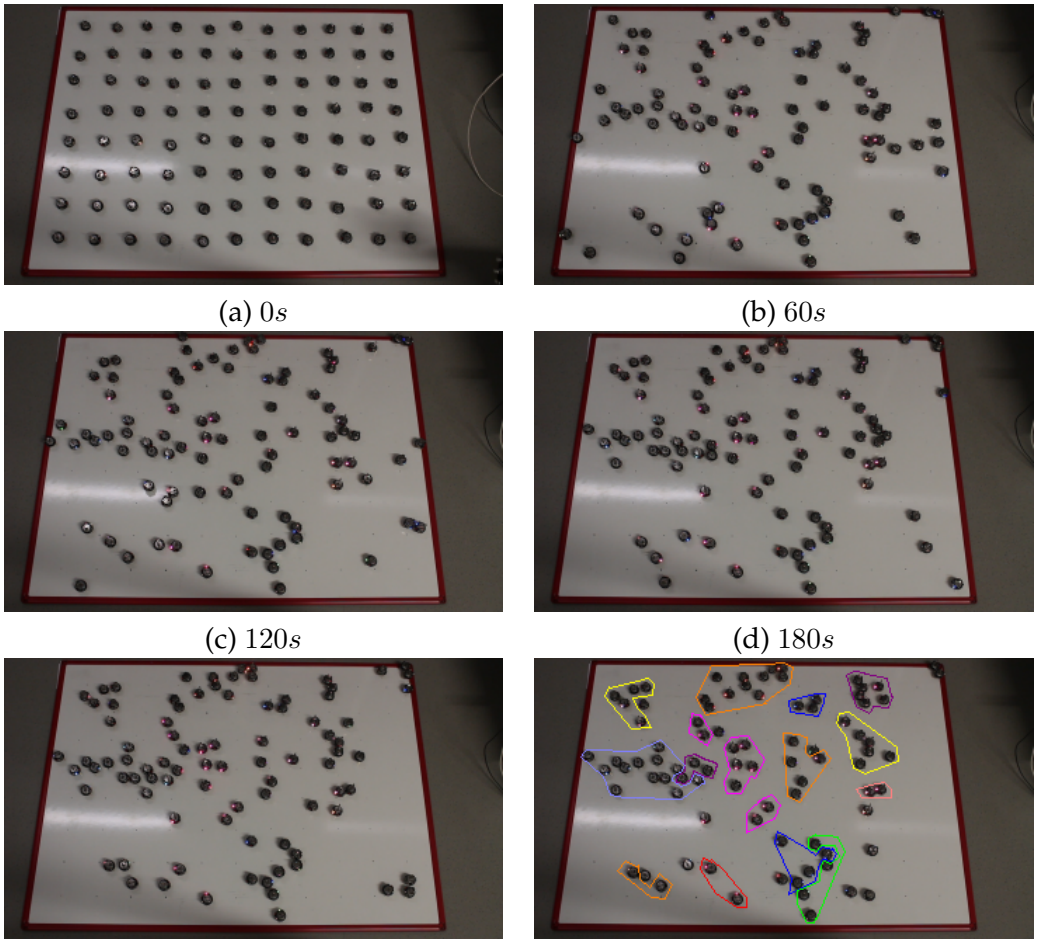



Figure 4.8: Mean dynamics of all 10 trials of the object clustering with 5 e-pucks and 20 objects. (a) and (c) are plotted using data from [49]. (b) and (d) present the results for the controller that was synthesised using SCT. In (a) and (b), the relationship between the proportion of clustered objects and time is displayed. (c) and (d) display the compactness of objects in relation to time. The thick red line indicates the mean, while the two brighter lines indicate the standard deviation.

Furthermore, a second set of experiments were performed to test the scalability of the approach. In this experiment 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) [38]. 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.

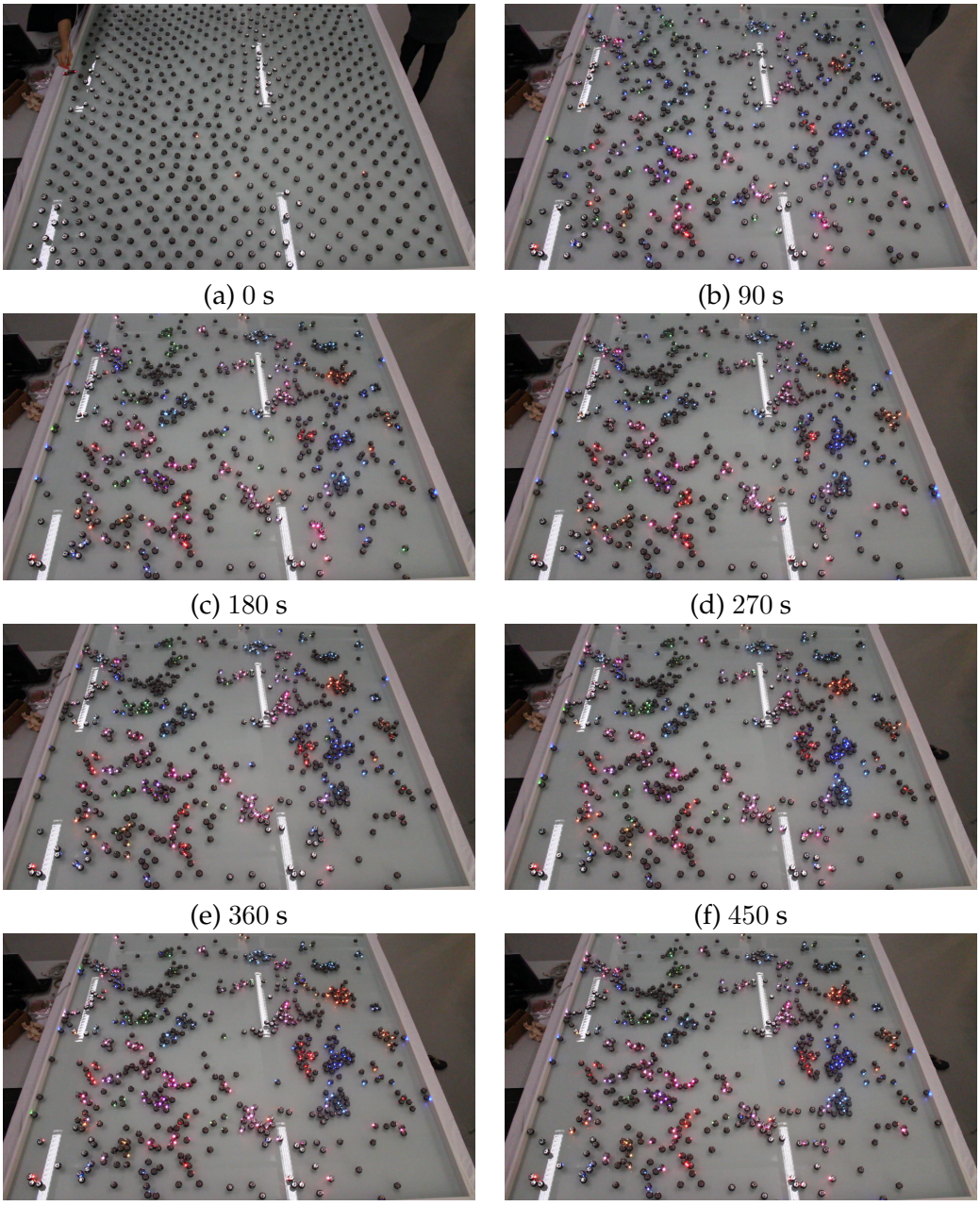
In total, 10 experimental trials were conducted. While it is difficult to monitor the con-



(a) $0s$ (b) $60s$ (c) $120s$ (d) $180s$ (e) $240s$ (f) $300s$

Figure 4.9: Sequence of snapshots taken from one of the trials where 88 Kilobots perform the group formation task. In (f) the formed groups are highlighted.

tinuous operation of 600 autonomous robots, visual inspection confirmed that they formed the groups as intended. Figure 4.10 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 on the online supplementary material [141]. Note that we use the robot's LED to 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 ($\pm 1$) 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.



(a) 0 s (b) 90 s

(c) 180 s (d) 270 s

(e) 360 s (f) 450 s

(g) 540 s (h) 600 s

Figure 4.10: Sequence of snapshots taken from one of the trials where 600 Kilobots perform the group formation task. Photo (a) shows the robots in their initial position, (b-g) show snapshots took at every 90 s showing the positions and states of robots, and (h) shows the robots at the end of the trial.

## 4.3 Summary

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 thesis 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, 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 [142] or may require alteration [86].

# 5

# Supervisory control of swarms of robots using global events

In Chapters 3 and 4 the SCT was applied to groups of robots in which each robot is treated as an isolated system. This means that all events related to a particular robot can only be observed by that robot. We name these types of events local events. Local events are sufficient to perform sensing and actuation of the robot's own hardware. However, robots may be required to communicate with other robots of the swarm to perform a common task [143]. Currently, the communication needs to be implemented case by case on a lower level of the SCT framework's control structure, the operational procedures.

In this chapter, we extend the SCT framework to incorporate the concept of global events in the context of swarm robotics. A global event is an event that is observable by the distributed system as a whole. This allows the formal definition of the communication between all robots of the swarm on the highest level of the SCT framework's control structure, the supervisor. We apply this novel concept to transparently implement communication among robots in a swarm through formally defined discrete events. This is demonstrated in four case studies, where we report experimental trials on a physical swarm of robots.

In Section 5.1, we introduce the novel concept of global events. The case studies are described in Section 5.2. Section 5.3 details the automatic code generation for the controllers supporting global events. Section 5.4 presents the experimental methods and results. The chapter is summarised in Section 5.5.

## 5.1 Supervisory control over global events

A swarm of robots can be seen as a large system compounded of multiple identical and independent smaller systems, the robots. These robots must cooperate to perform a task despite each robot executing its control autonomously. To cooperate, robots need to interact; in swarms of robots communication has been used to implement robots' interactions; for example, robots can detect the presence of a leader in the neighbourhood. The communication among robots using the SCT usually relies on a specific implementation not defined at the supervisor level.

Solutions for the integration of different systems using the SCT exist in the literature. Hierarchical approaches have been considered for the integration among systems. In [144], hierarchical interface-based supervisory control was applied in a flexible manufacturing system. In [145], a hierarchical SCT approach has been presented to integrate a simulated manufacturing cell with high level planning systems. Hierarchical interface-based supervisory control considers a master–slave relationship among systems.

Despite such a relationship being appropriate in the manufacturing context it goes against the concepts in practice in swarm robotics. In a manufacturing system there is a clear vertical structure with higher level systems, for example a scheduler, supervising low level systems, such as a manufacturing cell. In swarm robotics, such high level controllers are absent, as systems are horizontally related. Furthermore, hierarchical interface-based supervisory control enforces a serialization of requests (requests are made by the high-level subsystem; for each request the low-level sub-system must return an answer) [144].

To permit the formal specifications of communications horizontally among the robots of the swarm this thesis introduces the concept of global events. In this approach, the event set, $\Sigma$—usually formed by a set of controllable events, $\Sigma_c$, and a set of uncontrollable events, $\Sigma_u$—is split into four disjointed sets: *local uncontrollable events*, $\Sigma_{lu}$; *local controllable events*, $\Sigma_{lc}$; *global uncontrollable events*, $\Sigma_{gu}$; and *global controllable events*, $\Sigma_{gc}$. Where $\Sigma_c = \Sigma_{lc} \cup \Sigma_{gc}$ and $\Sigma_u = \Sigma_{lu} \cup \Sigma_{gu}$. The local events are equivalent to the traditional use of events in the SCT. When global events occur in a single robot, all other robots of the swarm are notified and use this information to evolve the current state of the supervisor.

As other robots only receive the information about the occurrence of global events on other robots, without being able to disable them, the only transmitted events are global uncontrollable ones. Therefore, each global controllable event requires a mapping to a global uncontrollable event. This map is incorporated in the definition of generator that supports global events, as:

$$G = (Q, \Sigma, \delta, q_0, Q_m, m), \tag{5.1}$$

where:

- $Q$ is a finite set of states;
- $\Sigma$ 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;
- $Q_m \subseteq Q$ is a set of marked states;
- and $m : (\Sigma_{gc} \cup \Sigma_{gu}) \rightarrow \Sigma_{gu}$ is the map from global events to the related global uncontrollable event to be transmitted.

In the next section we present a series of case studies using global events. In Section 5.2.1, we present the use of global uncontrollable events through a case study in swarm robotics. Case studies considering the map of controllable events to global uncontrollable events are shown in Sections 5.2.2, 5.2.3, and 5.2.4.

## 5.2 Modelling and supervisor synthesis

In the following, we provide guidance on how to model systems using the SCT with global events. We present four case studies that illustrate how it can be applied in swarm robotics. We use $\theta \in \{go, gl, gd, gs\}$ to refer to the different case studies applied to global events, where $go$ refers to to clustering objects in the presence of an intruder, $gl$ to last spotted location, $gd$ to disjoint agreement, and $gs$ to synchronous movement avoiding collision.

### 5.2.1 Case study one: clustering objects in the presence of an intruder

In this case study, we present an extension of the object clustering strategy [49]. Originally, the strategy allows a group of e-puck robots to cluster objects that are initially dispersed in the environment. The extended strategy establishes that the object clustering must only occur when a condition is met, in this case the presence of an intruder.

The swarm of robots performs a rotation over their central axis scanning for an intruder. Once one of the robot detects an intruder all robots begin to cluster the objects. If no intruder is detected by any robot for 30 s the robots return to their scanning mode.

Without global sensing, robots would cluster objects only when an intruder was locally detected. The sensing capabilities of the swarm would not be fully explored, leading to a partitioned swarm performing two different and likely conflicting behaviours simultaneously. The searching strategy would need to be comprehensive as each robot would rely on only its capabilities to find the intruder. With the proposed extension the entire swarm can benefit from information available to a single robot. As a result, the swarm performs the clustering algorithm together. This maximises the total time of robots performing the clustering and allows a simple approach for the searching behaviour.

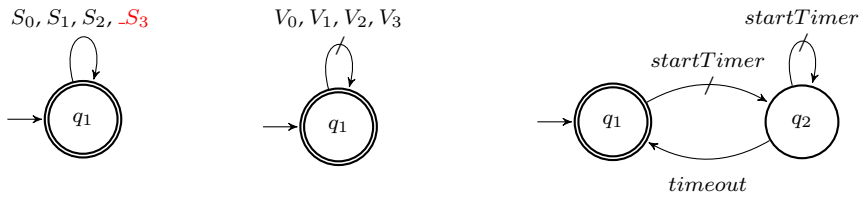
In the following, it is shown how global uncontrollable events enable the formalisation of this controller with the SCT. Figure 5.1 shows the free behaviour models for this strategy. Each robot $r_i$ with $i \in \{1, \cdots, R\}$ has its own set of free behaviour models. Free behaviour model $G_1^{go}$ represents the line-of-sight sensor. Uncontrollable events $S_0$, $S_1$, $S_2$, and $\_S_3$ represent, respectively, the presence of nothing, an object, another robot, or an intruder in the line-of-sight. Event $\_S_3$ is a global uncontrollable event and, therefore, its occurrence on a single robot will be vocalised for the entire swarm. Free behaviour model $G_2^{go}$ defines the possible movements. Controllable events $V_0$, $V_1$, $V_2$, and $V_3$ represent pairs $(v_{l0}, v_{r0})$, $(v_{l1}, v_{r1})$, $(v_{l2}, v_{r2})$, and $(v_{l3}, v_{r3})$, respectively. Where, $v_{li}$ and $v_{ri}$ are the left and right wheel velocities for $V_i$, respectively. The first three pairs were obtained using an evolutionary search [49]:

$$(v_{l0}, v_{r0}, v_{l1}, v_{r1}, v_{l2}, v_{r2}) = (0.5, 1, 1, 0.5, 0.1, 0.5), \tag{5.2}$$

The last pair was defined as rotating in place:

$$(v_{l3}, v_{r3}) = (0.3, -0.3). \tag{5.3}$$

Free behaviour model $G_3^{go}$ represents a timer, which triggers an event after 30 s of its activation. Controllable event $startTimer$ activates the timer. Once the time has elapsed uncontrollable event $timeout$ is triggered. In the case of $startTimer$ occurring during the 30 s interval, the timer is restarted. Therefore, $timeout$ will only be triggered after 30 s of the last timer's activation. In Table 5.1 the events' definition is summarised.




(a) $G_1^{go}$ (b) $G_2^{go}$ (c) $G_3^{go}$

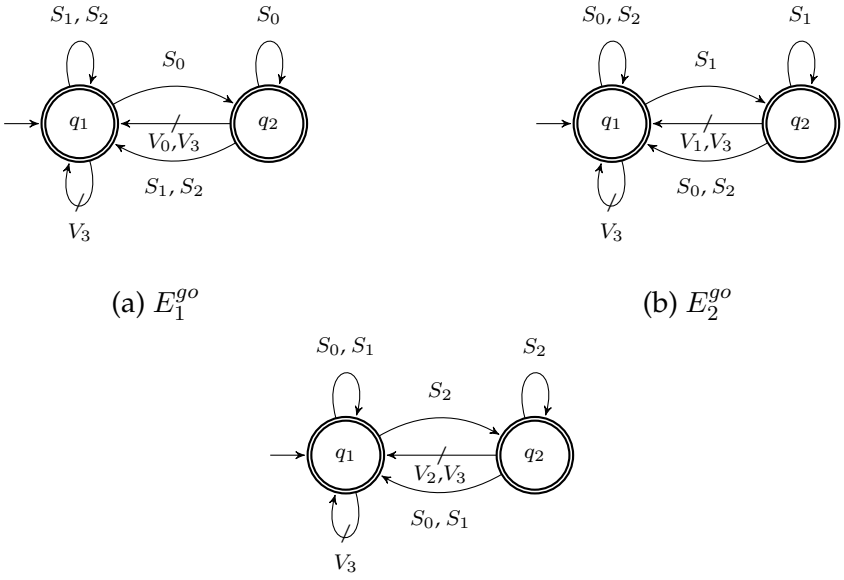
Figure 5.1: Free behaviour models for the clustering objects in the presence of an intruder case study. (a) Line-of-sight sensor to detect nothing, objects, robots, or an intruder; (b) motion capabilities; (c) timer. Global uncontrollable events are highlighted in red.

Table 5.1: Summary of events' definition for the cluster of objects in the presence of an intruder case study. In the controllability column local controllable events are indicated by *C*, local uncontrollable events are indicated by *U*, global controllable events are indicated by *GC*, and global uncontrollable events are indicated by *GU*.

| event | controllability | definition |
|---|---|---|
| $S_0$ | *U* | Binary sensor detects wall (or nothing) in its line-of-sight. |
| $S_1$ | *U* | Binary sensor detects an object in its line-of-sight. |
| $S_2$ | *U* | Binary sensor detects a robot in its line-of-sight. |
| $\_S_2$ | *GU* | Binary sensor detects an intruder in its line-of-sight. |
| $V_0$ | *C* | Robot's wheels speeds are set to $(v_{l0}, v_{r0}) = (0.5, 1)$. |
| $V_1$ | *C* | Robot's wheels speeds are set to $(v_{l1}, v_{r1}) = (1, 0.5)$. |
| $V_2$ | *C* | Robot's wheels speeds are set to $(v_{l1}, v_{r1}) = (0.1, 0.5)$. |
| $V_3$ | *C* | Robot's wheels speeds are set to $(v_{l1}, v_{r1}) = (0.3, -0.3)$. |
| $startTimer$ | *C* | Activates the timer. |
| $timeout$ | *U* | 30 s have elapsed since the timer's last activation. |

Figures 5.2, 5.3, and 5.4 illustrate the specifications for the object clustering strategy.

Specifications $E_1^{go}$, $E_2^{go}$, and $E_3^{go}$ relate, respectively, to the perception of nothing ($S_0$), an object ($S_1$), or another robot ($S_2$) with wheel velocities, which are specified by parameters $v_{li}$ and $v_{ri}$ through controllable events $V_i,\ i \in \{0, 1, 2, 3\}$. When receiving an event $S_i,\ i \in \{0, 1, 2\}$ the wheel velocities must be $V_i$ or $V_3$.




(a) $E_1^{go}$

(b) $E_2^{go}$

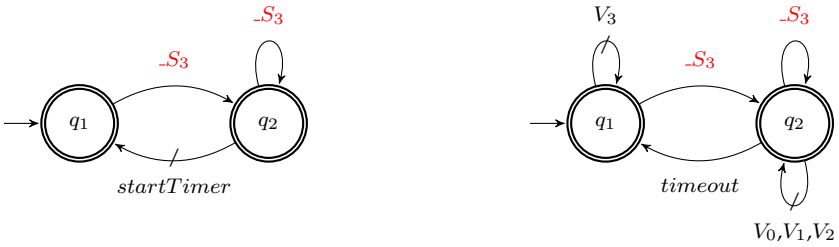
(c) $E_3^{go}$

Figure 5.2: Specifications regarding the robot's movement according to the value of the line-of-sight sensor reading for case study one.

Specification $E_4^{go}$ relates the perception of an intruder with the activation of the timer. Specification $E_5^{go}$ relates the perception of an intruder ($\_S_3$) in the last 30 s with the wheel velocities. If no intruder has been spotted—state $q_1$ in $E_5$—the robot's only option is to rotate in place (by triggering event $V_3$). Otherwise, only events $V_0$, $V_1$, and $V_2$ can be triggered.

Specifications $E_6^{go}$, $E_7^{go}$, $E_8^{go}$, and $E_9^{go}$ guarantee that any of the events $V_0$, $V_1$, $V_2$, and $V_3$ do not occur consecutively (e.g., when event $V_0$ occurs it cannot occur again until either $V_1$, $V_2$, or $V_3$ occur). The objective is to prevent the same movement activation from occurring consecutively (yet, the robot will perform its current movement indefinitely).

We use the local modular approach to synthesise the supervisors with support of global events. The operations used are the same for the traditional synthesis. Table 5.2 shows




(a) $E_4^{go}$ (b) $E_5^{go}$

Figure 5.3: Specifications regarding the intruder detection. (a) Activation of the timer when an intruder is detected; (b) the robot rotates on the spot if no intruder has been detected in the last 30 s.


(a) $E_6^{go}$ (b) $E_7^{go}$


(a) $E_8^{go}$ (b) $E_9^{go}$

Figure 5.4: Specifications preventing the same movement event from occurring consecutively (yet, the robot will perform its current movement indefinitely).

the relation of events for each specification for the cluster of objects in the presence of an intruder case study.

Table 5.2: Events used by the specifications and free behaviour models for the cluster of objects in the presence of an intruder case study.

| | | $E_1^{go}$ | $E_2^{go}$ | $E_3^{go}$ | $E_4^{go}$ | $E_5^{go}$ | $E_6^{go}$ | $E_7^{go}$ | $E_8^{go}$ | $E_9^{go}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $G_1^{go}$ | $s_0$ | ✓ | ✓ | ✓ | | | | | | |
| | $s_1$ | ✓ | ✓ | ✓ | | | | | | |
| | $s_2$ | ✓ | ✓ | ✓ | | | | | | |
| | $s_3$ | | | | ✓ | ✓ | | | | |
| $G_2^{go}$ | $v0$ | ✓ | | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| | $v1$ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| | $v2$ | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| | $v3$ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| $G_3^{go}$ | $startTime$ | | | | ✓ | | | | | |
| | $timeout$ | | | | | ✓ | | | | |
| local models | | $G_1^{loc,go}$ | $G_2^{loc,go}$ | $G_3^{loc,go}$ | $G_4^{loc,go}$ | $G_5^{loc,go}$ | $G_6^{loc,go}$ | $G_7^{loc,go}$ | $G_8^{loc,go}$ | $G_9^{loc,go}$ |

The local free behaviour models for the case study are obtained as:

$$\begin{array}{lll} G_1^{loc,go} = G_2^{loc,go} = G_3^{loc,go} & = & G_1^{go} || G_2^{go}, \\ G_4^{loc,go} & = & G_1^{go} || G_3^{go}, \\ G_5^{loc,go} & = & G_1^{go} || G_2^{go} || G_3^{go}, \\ G_6^{loc,go} = G_7^{loc,go} = G_8^{loc,go} = G_9^{loc,go} & = & G_2^{go}. \end{array} \tag{5.4}$$

The target language for each specification is obtained by:

$$K_i^{loc,go} = G_i^{loc,go} || E_i^{go} \quad \forall i \in \{1, \cdots, 9\}. \tag{5.5}$$

The local modular supervisors are:

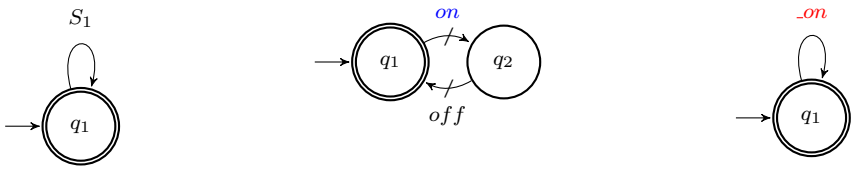
$$S_i^{loc,go} = SupC(G_i^{loc,go}, K_i^{loc,go}) \quad \forall i \in \{1, \cdots, 9\}. \tag{5.6}$$

### 5.2.2 Case study two: last spotted location

In this case study only the last robot in the swarm that spotted a red object must keep its Light-Emitting Diodes (LEDs) on. Figure 5.5 shows the free behaviour models for

this strategy. Each robot $r_i$ with $i \in \{1, \cdots, R\}$ has its own set of free behaviour models. Free behaviour model $G_1^{gl}$ represents the line-of-sight sensor. The uncontrollable event $S_1$ represents the presence of an object in the line-of-sight. Free behaviour model $G_2^{gl}$ defines the LEDs' status, controllable event $on$ and $off$ turns the LEDs on and off, respectively.

The use of communication among the robots makes the implementation of this single specification problem feasible. This is achieved by defining the event $on$ in free behaviour model $G_2^{gl}$ as global and by including free behaviour model $G_3^{gl}$. Free behaviour model $G_3^{gl}$ defines the global uncontrollable event $\_on$. The event $\_on$ (in $G_3^{gl}$) is mapped to the occurrence of the global controllable event $on$ (in $G_2^{gl}$) on other robots.




(a) $G_1^{gl}$ (b) $G_2^{gl}$ (c) $G_3^{gl}$

Figure 5.5: Free behaviour models for the last spotted location case study. (a) Line-of-sight sensor to detect an object; (b) The LED switch; (c) The LED activation of other robots, the global uncontrollable event $\_on$ is mapped to the occurrence of the global controllable event $on$ on other robots. Global uncontrollable events are highlighted in red and global controllable events are highlighted in blue.

In Table 5.3 the events' definition is summarised.

Table 5.3: Events' definition for the last spotted location case study.

| event | controllability | definition |
|---|---|---|
| $S_1$ | $U$ | Presence of an object in the line-of-sight. |
| $on$ | $GC$ | Turns the LEDs on. |
| $off$ | $C$ | Turns the LEDs off. |
| $\_on$ | $GU$ | Indicates the occurrence of $on$ in another robot. |

Figure 5.6 illustrates the single specification, $E_1^{gl}$, for this case. In state $q_1$ the robot can only turn off the LEDs (event $off$). If the robot's line-of-sight sensor detects an object (event $S_1$), the generator evolves to state $q_2$, where the robot can only turn on the LEDs, by triggering the global controllable event $on$. When the global controllable

event $on$ occurs in a particular robot, it triggers the occurrence of the respective global uncontrollable event, in this case, $\_on$. When $\_on$ occurs in all other robots, their state will be $q_1$, where the action of turning off the LEDs is enabled.

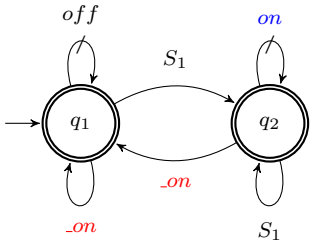



Figure 5.6: Specification $E_1^{gl}$ for the last spotted location case study.

As there is only a single specification that uses all free behaviour models, the modular, local modular, and monolithic supervisors are the same. The supervisor calculation follows the same procedure used in the previous case study and is therefore omitted.

### 5.2.3 Case study three: disjoint agreement

In this case study, two robots must perform a disjoint agreement regarding the selection of one of two options. The robots must select a different alternative from the one selected by the other robot. In this case, the options are to turn on the LEDs on the left or the right side. Figure 5.7 shows the free behaviour models for this strategy. Free behaviour model $G_1^{gd}$ represents a request for the left ($rL$) or right ($rR$) side. Free behaviour model $G_2^{gd}$ defines the global uncontrollable events $\_rL$ and $\_rR$. $\_rL$ and $\_rR$ are mapped to the occurrence of the global controllable events $rL$ and $rR$ (in $G_1^{gd}$) on other robots, respectively. Free behaviour model $G_3^{gd}$ defines the LEDs' status, controllable event $onLoffR$ turns on the LEDs from the left side and turns off the LEDs from the right side, controllable event $onRoffL$ does the opposite.

Free behaviour model $G_4^{gd}$ represents a timer, which triggers an event after 2 s of its activation. Controllable event $startTimer$ activates the timer. Once the time has elapsed, the uncontrollable event $timeout$ is triggered. In Table, 5.4 the events' definition is summarised.

Figure 5.8 illustrates the specifications. Specification $E_1^{gd}$ permits to turn on only the side that the robot had previously requested. Specification $E_2^{gd}$ does not allow to turn

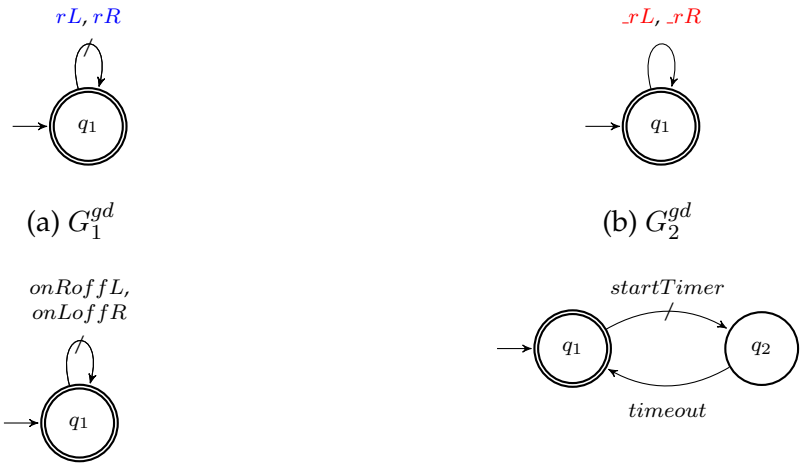
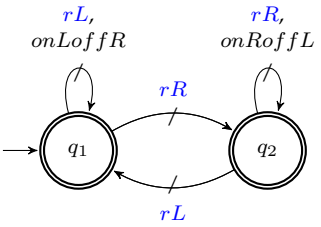



Figure 5.7: Free behaviour models for the disjoint agreement case study. (a) The ability of the robot to request left ($rL$) or right ($rR$); (b) Receiving other robot's suggestion through global uncontrollable events, the global uncontrollable events $\_rL$ and $\_rR$ are respectively mapped to the occurrence of the global controllable events $sL$ and $sR$ on other robots; (c) The LED activation of other robots; (d) robot's internal timer.

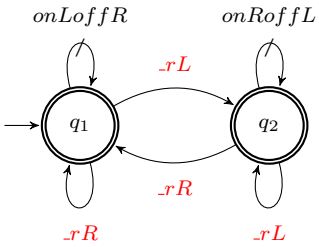
Table 5.4: Events' definition for the disjoint agreement case study.

| event | controllability | definition |
|---|---|---|
| $rL$ | *GC* | Request for the left side. |
| $rR$ | *GC* | Request for the right side. |
| $\_rL$ | *GU* | Indicate the request for left side in another robot. |
| $\_rR$ | *GU* | Indicate the request for right side in another robot. |
| $onRoffL$ | *C* | Turns on the LEDs from the right side and turns off the LEDs from the left side. |
| $onLoffL$ | *C* | Turns on the LEDs from the left side and turns off the LEDs from the right side. |
| $startTimer$ | *C* | Activates the timer. |
| $timeout$ | *U* | 2 s have elapsed since timer's activation. |

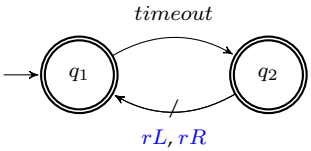
on the side that the other robot had previously requested. Specification $E_3^{gd}$ sets a request interval of 2 s. Specification $E_4^{gd}$ guarantees that the robot had made at least one request and received another request before turning on any side. Table 5.5 shows the relation of events for each specification.



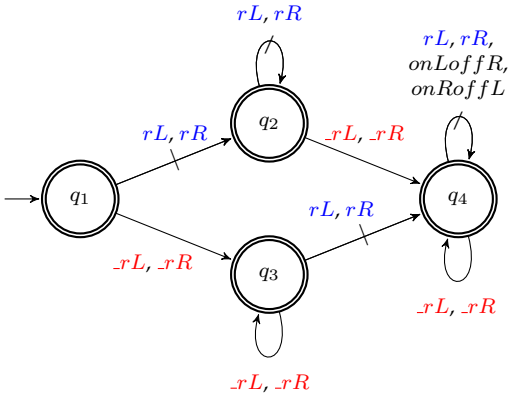

(a) $E_1^{gd}$




(b) $E_2^{gd}$




(c) $E_3^{gd}$




(d) $E_4^{gd}$

Figure 5.8: Specifications for the disjoint agreement case study. (a) Robots set the LEDs according to the requested they made. (b) Robots cannot turn on the LEDs requested by another robot. (c) The request takes place in a interval defined by the timer. (d) Any robot cannot turn on LEDs until both robots have made at least one request.

The local free behaviour models for the case study are obtained as:

$$\begin{aligned} G_1^{loc,gd} &= G_1^{gd}||G_3^{gd}, \\ G_2^{loc,gd} &= G_2^{gd}||G_3^{gd}, \\ G_3^{loc,gd} &= G_1^{gd}||G_4^{gd}, \\ G_4^{loc,gd} &= G_1^{gd}||G_2^{gd}||G_3^{gd}. \end{aligned} \tag{5.7}$$

Table 5.5: Events used by the specifications and free behaviour models for the disjoint agreement case study.

| | | $E_1^{gd}$ | $E_2^{gd}$ | $E_3^{gd}$ | $E_4^{gd}$ |
|---|---|---|---|---|---|
| $G_1^{gd}$ | $rL$ | ✓ | | ✓ | ✓ |
| | $rR$ | ✓ | | ✓ | ✓ |
| $G_2^{gd}$ | $\_rL$ | | ✓ | | ✓ |
| | $\_rR$ | | ✓ | | ✓ |
| $G_3^{gd}$ | $onRoffL$ | ✓ | ✓ | | ✓ |
| | $onLoffR$ | ✓ | ✓ | | ✓ |
| $G_4^{gd}$ | $startTime$ | | | | |
| | $timeout$ | | | ✓ | |
| local models | | $G_1^{loc,gd}$ | $G_2^{loc,gd}$ | $G_3^{loc,gd}$ | $G_4^{loc,gd}$ |

The target language for each specification is obtained by:

$$K_i^{loc,gd} = G_i^{loc,gd} || E_i^{gd} \quad \forall i \in \{1, \cdots, 4\}. \tag{5.8}$$

The local modular supervisors are:

$$S_i^{loc,gd} = SupC(G_i^{loc,gd}, K_i^{loc,gd}) \quad \forall i \in \{1, \cdots, 4\}. \tag{5.9}$$

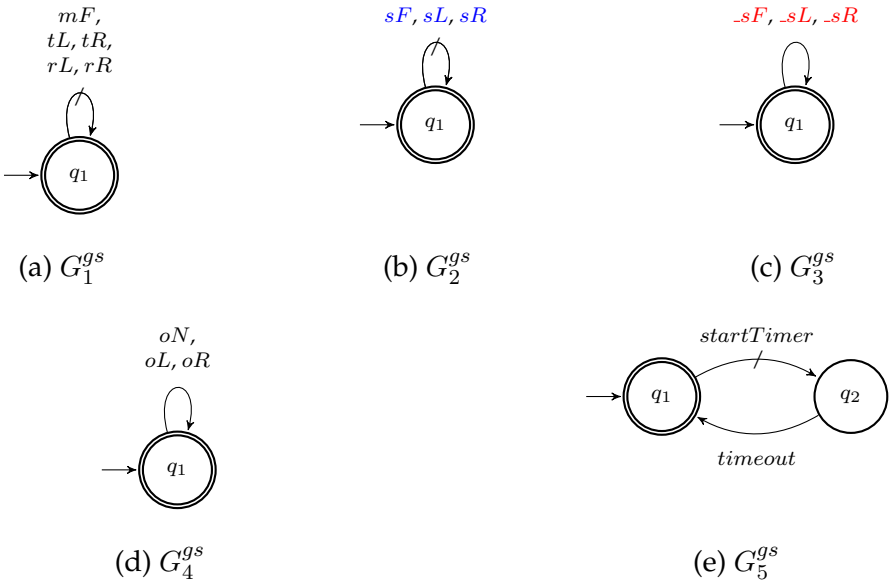
### 5.2.4 Case study four: synchronous movement avoiding collision

In this case study, robots must perform a synchronised movement while avoiding collision with the surrounding arena's wall and among each other. Robots can perform five of the following movements. Robots that are not avoiding collision must agree in performing the same one of three basic movements: moving forward, turning left, or turning right. Robots that detect the presence of an obstacle do not need to perform the same movement of the swarm and instead will perform one of the two movements for collision avoidance: rotating left or rotating right.

Figure 5.9 shows the free behaviour models for this strategy. Free behaviour model $G_1^{gs}$ defines the movements that the robots can perform. Controllable events $mF$, $tL$, $tR$, $rL$, and $rR$ instruct the robots to start moving forward, turning left, turning right, rotating left, or rotating right, respectively. Free behaviour model $G_2^{gs}$ represents the request for

one of the basic movements: $sF$ for moving forward, $sL$ for turning left, and $sR$ for turning right. Free behaviour model $G_3^{gs}$ defines the global uncontrollable events $\_sF$, $\_sL$, and $\_sR$. $\_sF$, $\_sL$, and $\_sR$ are mapped to the occurrence of the global controllable events $sF$, $sL$, and $sR$ of $G_2^{gs}$ in other robots, respectively.

Free behaviour model $G_4^{gs}$ defines the proximity sensor that detects the presence of objects (robots or walls). Uncontrollable events $oL$ and $oR$ occur if there was an object detected on the left or right side during the last second, uncontrollable event $oN$ is triggered otherwise. Free behaviour model $G_5^{gs}$ represents a timer, which triggers an event after 10 s of its activation. Controllable event $startTimer$ activates the timer. Once the time has elapsed uncontrollable event $timeout$ is triggered. In Table 5.6 the events' definition is summarised.




(a) $G_1^{gs}$ (b) $G_2^{gs}$ (c) $G_3^{gs}$

(d) $G_4^{gs}$ (e) $G_5^{gs}$

Figure 5.9: Free behaviour models for the synchronous movement case study.

Figure 5.10 illustrates the specifications for this case. Specifications $E_1^{gs}$, $E_2^{gs}$, and $E_3^{gs}$ start the robot movement according to the movement requested or received from another robot. Each of the three first specifications are concerned with a particular movement. When a robot requests to move forward, event $sF$, the state of specifications $E_1^{gs}$; $E_2^{gs}$; and $E_3^{gs}$ are $q_2$; $q_1$; and $q_1$, respectively, which only allow the controller to trigger the move forward event, $mF$. As $sF$ is mapped to $\_sF$, all other robots will also have the same state configuration, causing the swarm to perform the same movement.

Table 5.6: Events' definition for the synchronous movement case study.

| event | controllability | definition |
|---|---|---|
| $mF$ | $C$ | Instruct the robots to start moving forward. |
| $tL$ | $C$ | Instruct the robots to start turning left. |
| $tR$ | $C$ | Instruct the robots to start turning right. |
| $rL$ | $C$ | Instruct the robots to start rotating left. |
| $rF$ | $C$ | Instruct the robots to start rotating right. |
| $sF$ | $GC$ | Request for moving forwards. |
| $sL$ | $GC$ | Request for turning left. |
| $sR$ | $GC$ | Request for turning right. |
| $\_sF$ | $GU$ | Indicate the request for moving forwards in another robot. |
| $\_sL$ | $GU$ | Indicate the request for turning left in another robot. |
| $\_sR$ | $GU$ | Indicate the request for turning right in another robot. |
| $oN$ | $U$ | No object has been detected. |
| $oL$ | $U$ | Object detected on the left side. |
| $oR$ | $U$ | Object detected on the right side. |
| $startTimer$ | $C$ | Activate the timer. |
| $timeout$ | $U$ | 2 s have elapsed since timer's activation. |

Specification $E_4^{gs}$, forbids the three basic movements if an obstacle is detected and only allows the robot to rotate to the opposite direction to that of the object. Specification $E_5^{gs}$ sets a request interval of 10 s. Table 5.7 shows the relation of events for each specification for the synchronous movements avoiding collision case study.

The local free behaviour models for the case study are obtained as:

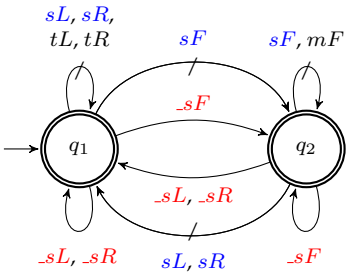
$$\begin{array}{lcl} G_1^{loc,gs} = G_2^{loc,gs} = G_3^{loc,gs} & = & G_1^{gs}||G_2^{gs}||G_3^{gs}, \\ G_4^{loc,gs} & = & G_1^{gs}||G_4^{gs}, \\ G_5^{loc,gs} & = & G_2^{gs}||G_5^{gs}. \end{array} \tag{5.10}$$

The target language for each specification is obtained by:

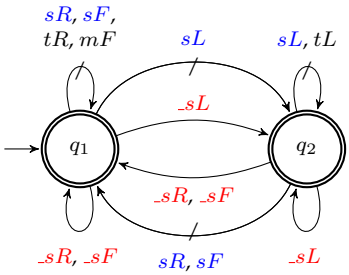
$$K_i^{loc,gs} = G_i^{loc,gs}||E_i^{gs} \quad \forall i \in \{1, \cdots, 5\}. \tag{5.11}$$

The local modular supervisors are:

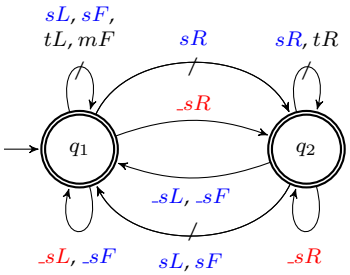
$$S_i^{loc,gs} = SupC(G_i^{loc,gs}, K_i^{loc,gs}) \quad \forall i \in \{1, \cdots, 5\}. \tag{5.12}$$



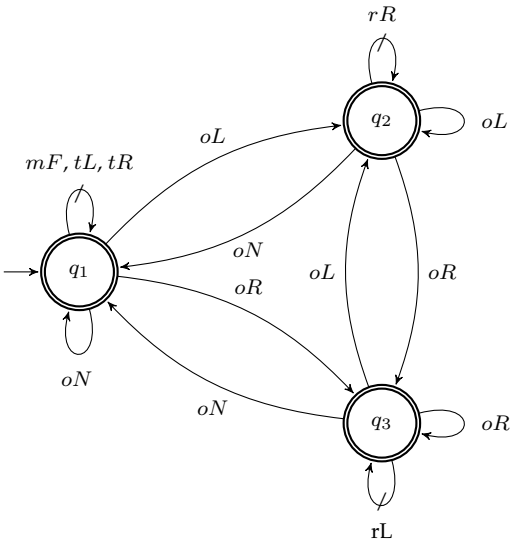

(a) $E_1^{gs}$



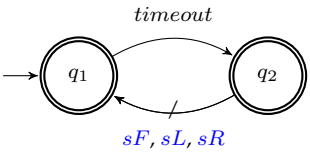

(b) $E_2^{gs}$




(c) $E_3^{gs}$




(d) $E_4^{gs}$




(e) $E_5^{gs}$

Figure 5.10: Specifications for the synchronous movement case study.

Table 5.7: Events used by the specifications and free behaviour models for the synchronous movement avoiding collision case study.

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

### 5.2.5 Comparison

Table 5.8 compares the three methods for supervisor synthesis in terms of the size of the target language and supervisors for all case studies with global events.

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 local modular approach turned out to be the most memory efficient in all case studies. The modularity cannot be used in case study two as it has a single specification.

## 5.3 Implementation

This section details the implementation of the supervisors containing global events. This is achieved by modifying the generator player—also called automata player—presented in Chapter 4 to communicate the occurrence of global events transparently. Apart from flagging the desired event as global, and defining a map from each global

Table 5.8: Total number of states and transitions for each case study when using monolithic, modular, and local modular synthesis approaches, respectively. Data corresponds to the target language $K$ and supervisor $S$. The best results are highlighted in bold.

| | | Monolithic | | Modular | | Local modular | |
|---|---|---|---|---|---|---|---|
| | | K | S | K | S | K | S |
| Case one | states | 102 | 85 | 36 | 35 | **22** | **21** |
| | transitions | 568 | 470 | 317 | 310 | **122** | **115** |
| Case two | states | 4 | 4 | 4 | 4 | 4 | 4 |
| | transitions | 10 | 10 | 10 | 10 | 10 | 10 |
| Case three | states | 36 | 27 | 20 | 19 | **12** | **11** |
| | transitions | 143 | 96 | 115 | 108 | **37** | **34** |
| Case four | states | 44 | 44 | 22 | 22 | **13** | **13** |
| | transitions | 420 | 420 | 286 | 286 | **93** | **93** |

controllable event to a global uncontrollable event, there is no another action required by the user of this approach.

The generator player is a virtual machine. It executes the generators which realise the supervisors. We extend the open source software tool Nadzoru [146, 1, 27] to support global events in the modelling of free behaviour models and specifications in the supervisor synthesis, and in the automatic generation of the controller's source code. A communication protocol and router software to deal with the required network communication were developed.

### 5.3.1 Communication

The e-puck [37] uses a Bluetooth module for wireless communication. Bluetooth is defined in IEEE 802.15.1 standard [147] on a $2.4-2.485\ GHz$ band. It is a package-based master/slave protocol. A single master can communicate with up to seven slaves via point-to-point connections in a network called piconet.

Our controller uses the Bluetooth capabilities of the e-puck to establish a wireless network. We implemented a router to act as the central hub of a star topology, as shown in Figure 5.11. Using the ISO OSI model [148], the proposed controller operates on top of underlying network application layer and, consequently, operates independently from the network topology. In other words, the controller applies to other network topologies.

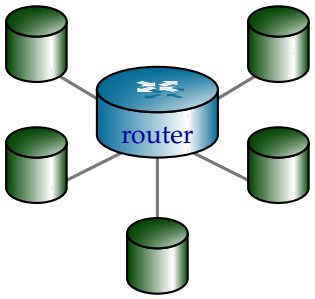



Figure 5.11: Star topology used for the communication between robots. The e-puck robots are in green and the router is in blue.

Each device maintains a data buffer for each connection to another device. Messages are buffered before being sent via Bluetooth. Messages are removed from the buffer once an acknowledgement from the recipient is received or a predefined number of attempts have been made. The robot is only required to have a single buffer, whereas the router keeps one buffer for each connection. This approach, therefore, moves the memory intensive message buffering from e-pucks to the router, which is implemented on a desktop computer.

We implemented a communication protocol that handles the transmission of data through the serial data service over Bluetooth. The package is composed by a header that contains information regarding the package router and encapsulates a variable length data section. The data section contains the information required to implement distributed sensing using global uncontrollable events.

Figure 5.12 illustrates the structure of the package, which is composed of the following: (1) the start byte, (2) the size of the data payload (#pl, i.e., the size in bytes of the data being transmitted), (3) an acknowledgement identification (ack), (4) the checksum value (cks), (5) the command (cmd), (6) the command parameters (cpr), (7) the data, and (8) the end byte.

Whenever the router receives a message from a sender robot, it reads the package's header and checks the package's consistency by using a checksum algorithm. If the package is valid, the router emits an acknowledgement message to the sender robot and performs the required action defined in the command. The command (cmd) component of the package allows different actions, one of which is *broadcasting*, where all robots receive a copy of the message. To perform the broadcast, the router queues one copy of the message to be transmitted in the buffers for the recipient robots.

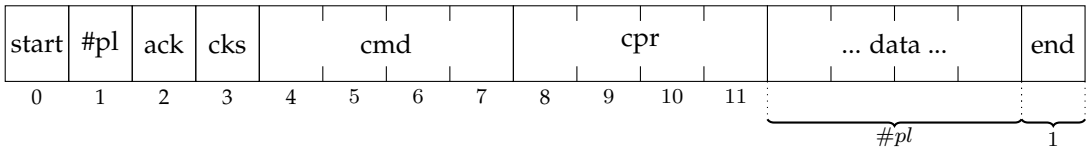



Figure 5.12: Data structure of the package used to transmit the occurrence of global events. The structure allows the definition of a command and a variable payload of up to 255 bytes.

Messages in the buffer are transmitted to the respective robot. To account for any transmission error the router waits for an acknowledgement message. If the acknowledgement is not received within $T^{ack}$ ms another attempt is made with an increased $T^{ack}$. The value of $T^{ack}$ is $T_i^{ack} = 2000 + 1000 \times i^{1.41}$ ms, where $i$ is the number of the attempt—for the initial attempt $i = 0$. After seven failed attempts, the message is discarded[1]. If the acknowledgement is received the message is removed from the buffer. Robot and router use the same acknowledgement strategy.

### 5.3.2 Memory representation

The memory representation used for global events is an extension of the one presented on Section 4.1.1. The extension consists of the addition of two vectors of size $e$ bytes; one to store whether events are global or local events and another to define the map between global controllable events and global uncontrollable events.

### 5.3.3 Generator player

We modified the original generator player [56] to support global events. The modified version is given by Algorithm 2. Its logic stores the states of all generators. First, it checks if any global uncontrollable event has occurred on another robot (see Algorithm 2 line 8). Then, it checks if uncontrollable events occurred locally—either global, in line 13, or local, in line 19—by calling functions in the operational procedures (see Section 5.3.4). If the uncontrollable event that occurred locally is a global event (lines 13-18) then the swarm is notified (line 18). This is achieved by broadcasting a message

[1] This usually means a more serious problem such as a link disconnection or the discharge of the robot's battery. In such a case the experiment would be considered to fail; this, however, did not occur during our experiments.

using the communication protocol previously described. If any uncontrollable event occurred (local or global), the generators' states are updated accordingly (lines 9-11, 14-16, and 20-22). Otherwise, the generator player determines the list of enabled controllable events (line 25). If the list is not empty, the generator player selects one event, at random, and updates the generators' states accordingly (lines 26-30). It also calls functions in the operational procedures to perform the action associated with the event (line 31). If the controllable event that occurred is a global event (lines 33-34) then the swarm is notified (line 33) with the respective global uncontrollable event that is mapped by the $Map$ function.

Note that the generator player (controller) prioritises the updating of the current state based on the occurrence of uncontrollable events over the generation of a controllable event. This is important as the control decision must be taken based on the most recent state that is available. Also, the generator player prioritises global uncontrollable events over local uncontrollable events as global uncontrollable events affect the entire system.

### 5.3.4 Operational procedures

The operational procedures interface the controller to the hardware [22]. They define one callback function per event. Unlike the control logic, these functions are not automatically derived from the formal specifications. For each event that is triggered the generator player calls this function to perform some action (see lines 12,17,23 and 31 of Algorithm 2). The operational procedures implement the routines to determine whether an uncontrollable event has occurred locally on the robot (see lines 13 and 19 of Algorithm 2).

The use of global events eliminates the use of the operational procedures to implement swarm-wide communication, keeping the same method for the use of the operational procedures for the definition of other actuation and sensing.

## 5.4 Experiments

This section describes the experiments that we performed using local modular supervisors that make use of global events. The experiments are used to validate our implementation of SCT in practice. The electronic supplementary material offers a selection

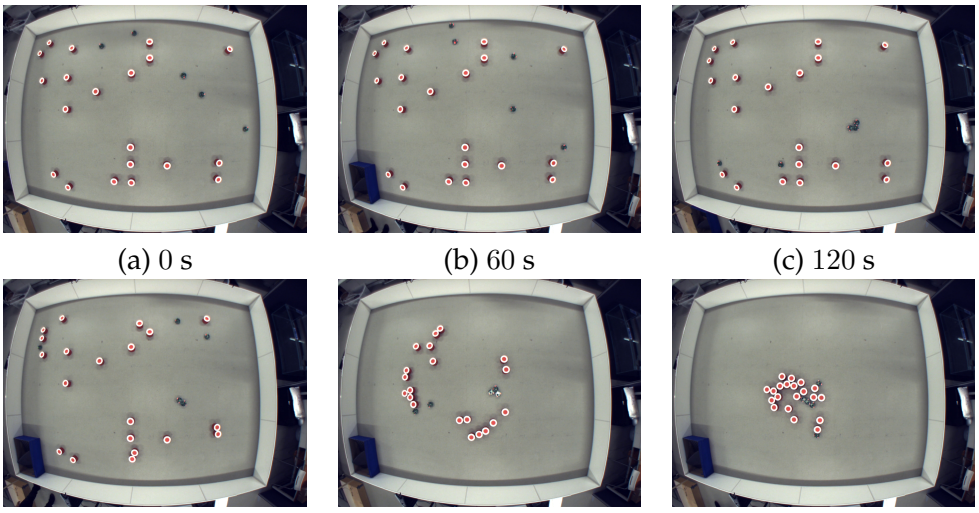
**Algorithm 2** Generator player

Let $N$ be the number of generators
Let $Map : \Sigma_{gc} \rightarrow \Sigma_{gu}$ be the map from all global controllable events to global uncontrollable events.
**procedure** GENERATOR PLAYER
    **for all** $j \in \{1, 2, \ldots, N\}$ **do**
        set current state $c_j$ to initial state $q_{0_j}$;
    **end for**
    **while** *true* **do**
        **if** $e_{gu} \in \Sigma_{gu}$ occurred globally **then**
            **for all** $j \in \{1, 2, \ldots, N\}$ **do**
                $c_j = \delta_j(c_j, e_{gu})$;
            **end for**
            execute callback function of $e_{gu}$;
        **else if** $e_{gu} \in \Sigma_{gu}$ occurred locally **then**
            **for all** $j \in \{1, 2, \ldots, N\}$ **do**
                $c_j = \delta_j(c_j, e_{gu})$;
            **end for**
            execute callback function of $e_{gu}$;
            Transmit( $e_{gu}$ );
        **else if** $e_u \in \Sigma_{lu}$ occurred locally **then**
            **for all** $j \in \{1, 2, \ldots, N\}$ **do**
                $c_j = \delta_j(c_j, e_u)$;
            **end for**
            execute callback function of $e_u$;
        **else**
            calculate the set of enabled controllable events $\psi(c_1, c_2, \ldots, c_n) : c_x \in \Sigma_c$;
            **if** $\psi(c_1, c_2, \ldots, c_n) \neq \emptyset$ **then**
                randomly select $e_c \in \psi(c_1, c_2, \ldots, c_n)$;
                **for all** $j \in \{1, 2, \ldots, N\}$ **do**
                    $c_j = \delta_j(c_j, e_c)$;
                **end for**
                execute callback function of $e_c$;
                **if** $e_c \in \Sigma_{gc}$ **then**
                    Transmit( $Map(e_{gc})$ );
                **end if**
            **end if**
        **end if**
    **end while**
**end procedure**

of video recordings. Video recordings from all 40 experimental trials and additional resources (models, the Nadzoru tool, the used source code) can be found on the online supplementary material [141].

### 5.4.1 Case study one: clustering objects in the presence of an intruder

We performed experiments with object clustering in the presence of an intruder with a controller automatically generated from the supervisor obtained using the SCT framework. The e-puck robot uses its on-board camera to implement the line-of-sight sensor, as described in Sector 4.2.3.



(a) 0 s (b) 60 s (c) 120 s

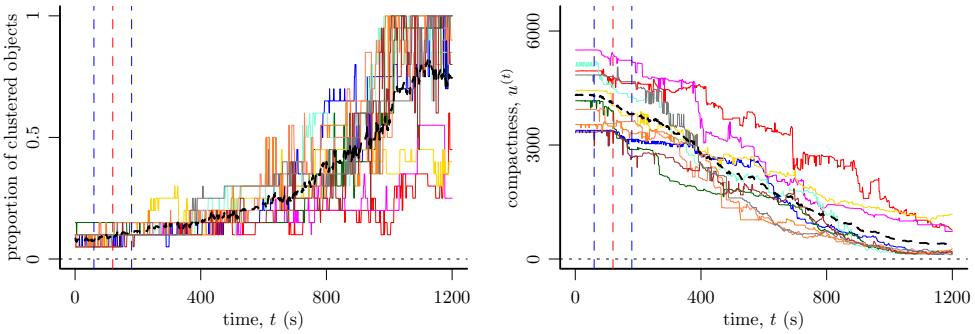
(d) 180 s (e) 600 s (f) 1200 s

Figure 5.13: The sequence of snapshots taken from one of the ten trials in which five e-pucks—in black—cluster 20 objects—in red—in the presence of an intruder—in blue. Photo (a) displays the initial positions of the robots and the objects. Photo (b) shows the placed intruder, which was removed after 60 s (Photo (c)). Photos (d-f) illustrate permanently placed intruder (180 s), the distribution of objects after a run-time of 600 s and 1200 s.

To simplify identification, the robots were fitted with green skirts and the objects to cluster used were red cylinders with 10 cm diameter and 10 cm height. The intruder object was a 43 cm $\times$ 63 cm blue box with 35 cm height. Trials were performed in a 400 cm $\times$ 300 cm light-grey-floor arena surrounded by white walls that were 50 cm in height. The arena had 165 pencil marks distributed as a 15 $\times$ 11 grid with columns and rows spaced 25 cm apart. Five e-pucks and 20 objects were uniformly randomly

distributed over the marks in the arena. A randomly distributed orientation for each robot was also assigned.

Ten trials were performed, each lasting 1200 s. An infrared (IR) signal was emitted to instruct the robots to start the controller. After 60 s the intruder was inserted in the corner of the arena, after a further 60 s the intruder was removed and finally after another 60 s, reinserted again in the same corner. In the case of the robots not being able to detect the intruder for an interval bigger than 120 s from the start of the searching behaviour, the intruder is moved to the next free corner in the clock-wise direction. In the case of a failure of any robot (e.g., the robot reset because of a collision or low battery, or a robot gets stuck on the arena or wall for a period greater than 60 s), a restart of the controller was attempted via IR signal. Figure 5.13 shows snapshots taken from one of the experimental trials.

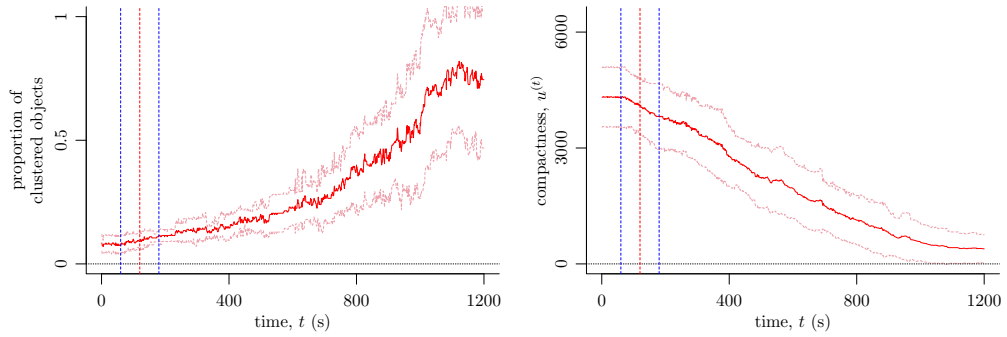



(a) (b)

Figure 5.14: Dynamics of object clustering with intruder detection with five e-pucks and 20 objects. In (a), the relationship between the proportion of clustered objects and time is displayed. In (b), 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. The vertical dashed lines represent the introduction of the intruder (in blue) and its remotion (in red).

To evaluate the performance of the controller we measured the proportion of objects in the largest cluster and the compactness of objects as defined in [49]. The clustering dynamics are plotted in Figure 5.14. The coloured curves correspond to the ten trials; the black dashed line represents the mean. Figure 5.14(a) shows the proportion of objects in the largest cluster. Figure 5.14(b) shows the compactness of objects.

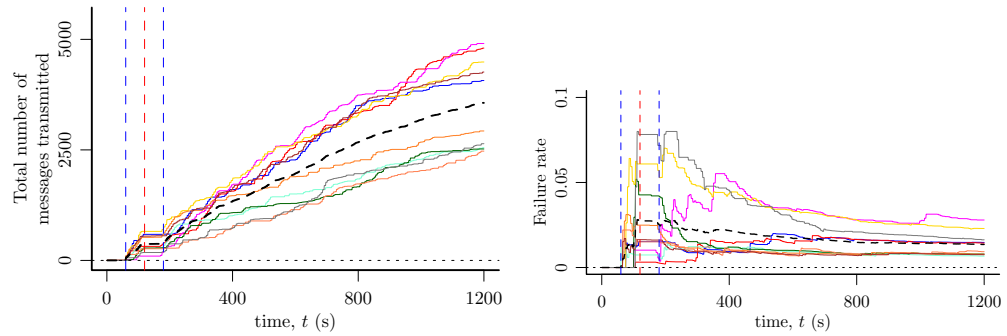
To evaluate the reliability of the communication, the router collected statistics about




(a) (b)

Figure 5.15: Mean dynamics of all ten trials. (a) and (b) plot the proportion of clustered objects and display the compactness of objects over time, respectively. The thick red line indicates the mean, while the two lighter lines indicate the standard deviation.

the transmission of packages. Figure 5.16(a) shows the accumulated number of message transmission attempts by the router to the robots. The vertical lines indicate when the intruder is introduced (in blue) or removed (in red). Figure 5.16(b) shows the accumulated failure rate of the message against the total of attempted messages.

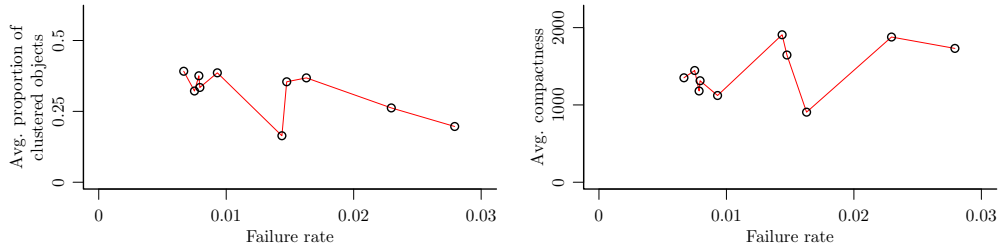



(a) (b)

Figure 5.16: (a) The number of message transmission attempts by the router to the robots. (b) Failure rate.

The reliability of the Bluetooth communication may impact the performance of the swarm behaviour. In Figure 5.17(a) the failure rate (see Figure 5.16(b)) is related to the average proportion of clustered objects. Trials with a bigger failure rate seem to tend to have a slower convergence rate in the number of clustered objects. The failure rate also seems to impact on the convergence rate based on the average compactness, as shown in Figure 5.17(b). Though, given the small sample size, the effects may not be

significant.




(a) (b)

Figure 5.17: Impact of the Bluetooth communication performance on the swarm behaviour. (a) Average proportion of clustered objects against the failure rate, (b) the average compactness against the failure rate.

### 5.4.2 Case study two: last spotted location

Experiments were performed with the last location identification strategy with a controller automatically generated from the supervisor obtained using the SCT framework. Trials were performed in a 71 cm long $\times$ 41 cm wide white floor arena surrounded by white walls that were 8 cm in height. Three e-puck robots were positioned equally spaced on a centred line parallel to the length. All robots were oriented to the same wall. Ten trials were performed, each lasting up to 60 s. An IR signal was emitted to instruct the robots to start the controller. An object was moved 25 times in front of one of the robots in a random order. Figure 5.18 shows snapshots taken from one of the experimental trials.

To evaluate the performance of the communication, the router collected statistics about the transmission of packages. Figure 5.19 shows the accumulated number of message transmission attempts by the router to the robots. During all the trials there was no message transition failure.

### 5.4.3 Case study three: disjoint agreement

We performed experiments with the disjoint agreement case study with a controller automatically generated from the supervisor obtained using the SCT framework. Trials were performed in a 71 cm long $\times$ 41 cm wide white floor arena surrounded by white

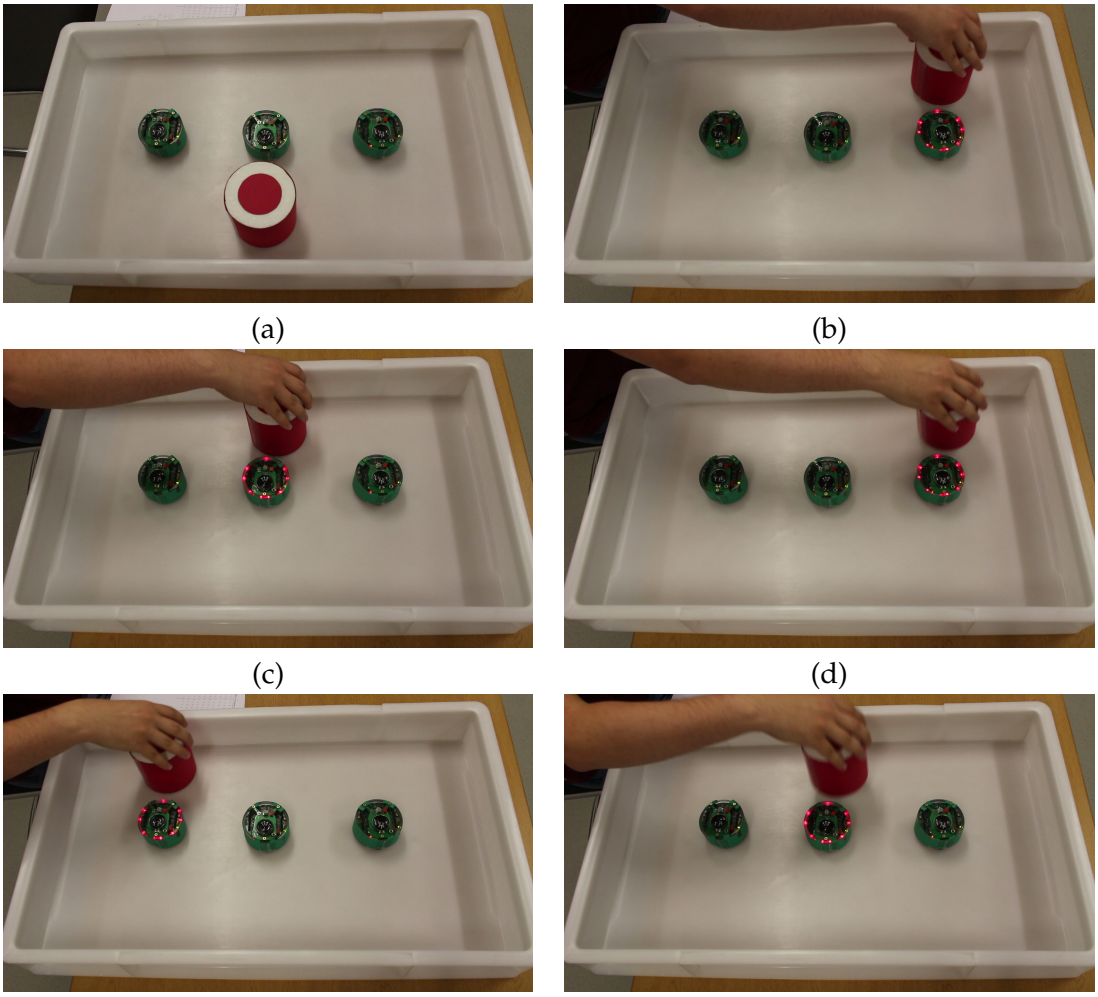


Figure 5.18: The sequence of snapshots taken from one of the ten trials in which three e-pucks perform last spotted location case study. Photo (a) displays the positions of the robots. Photos (b-f) show the first 5 changes.

walls that were 8 cm in height. Two e-puck robots were positioned in a centred line parallel to the length. All robots were oriented to the same wall. Ten trials were performed, each lasting up to 300 s. An IR signal was emitted to instruct the robots to start the controller. Figure 5.20 shows snapshots taken from one of the experimental trials.

To evaluate the performance of the communication, the router collected statistics about the transmission of packages. Figure 5.21(a) shows the accumulated number of message transmission attempts by the router to the robots. Figure 5.21(b) shows the accumulated failure rate of the message against the total of attempted messages.

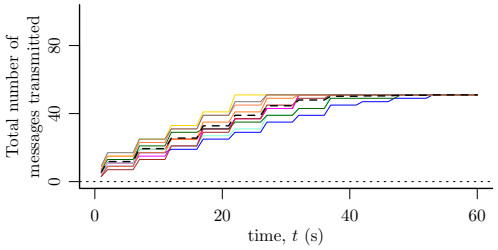



Figure 5.19: (a) The number of message transmission attempts by the router to the robots.

### 5.4.4 Case study four: synchronous movement avoiding collision

Trials were performed in a $400$ cm $\times$ $300$ cm light grey floor arena surrounded by white walls that were $50$ cm in height. The arena had $165$ pencil marks distributed as a $15 \times 11$ grid with columns and rows spaced $25$ cm apart. Five e-pucks were uniformly randomly distributed over the marks in the arena. A randomly distributed orientation for each robot was also assigned. Ten trials were performed, each lasting $300$ s. An IR signal was emitted to instruct the robots to start the controller.

Figure 5.22 shows snapshots taken from one of the experimental trials in which five e-pucks perform synchronous movements avoiding collision. The $9$ snapshots are superimposed to show how the robots are synchronised, the first snapshot is the most transparent one and the last the most opaque.

To evaluate the performance of the communication, the router collected statistics about the transmission of packages. Figure 5.23(a) shows the accumulated number of message transmission attempts by the router to the robots. Figure 5.23(b) shows the accumulated failure rate of the message against the total of attempted messages.

## 5.5 Summary

In this chapter, the use of global events for the supervisory control of swarm of robots was proposed. Global events are shared among all the robots of the swarm, abstracting the communication between them. The transmitted global events are always uncontrollable events. If a controllable event must be transmitted, a map to uncontrollable events is used. This approach allows the formal definition of communication among robots, eliminating the need to implement communication in the operational procedures.

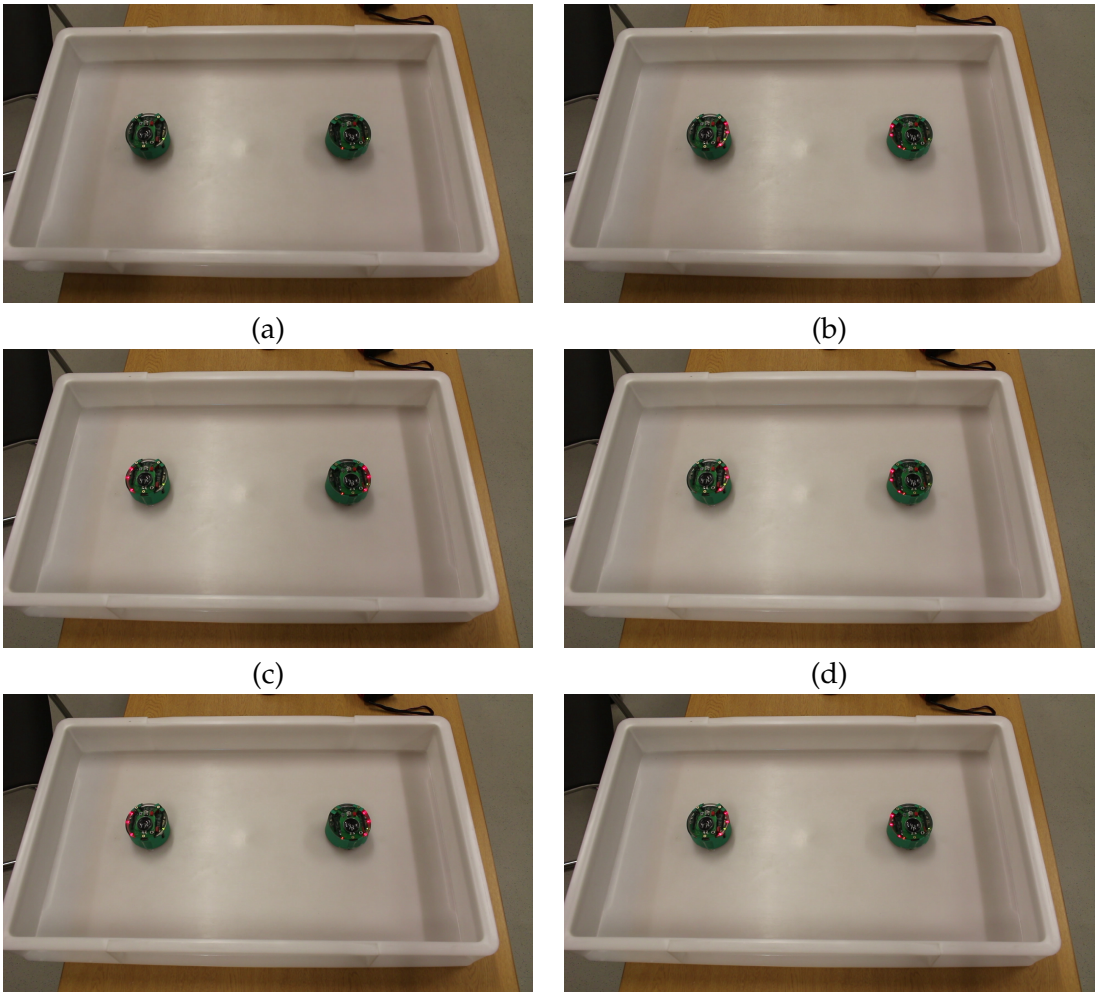
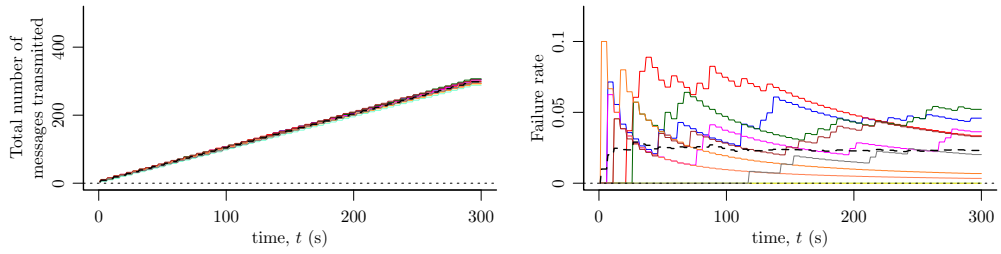


Figure 5.20: The sequence of snapshots taken from one of the ten trials in which two e-pucks performs the disjoint agreement case study. Photo (a) displays the positions of the robots. Photos (b-f) show the first 5 changes.

To illustrate the proposed framework, four case studies were presented. In the first case study, only global uncontrollable events are used in conjunction with local controllable and local uncontrollable events. In the other three case studies global controllable events are also considered. We showed that communications strategies can be modelled successfully with the use of global events within the extended SCT framework. Each of the case studies presented the expected behaviour as defined by the specifications.




(a) (b)

Figure 5.21: (a) The number of message transmission attempts by the router to the robots. (b) Failure rate.

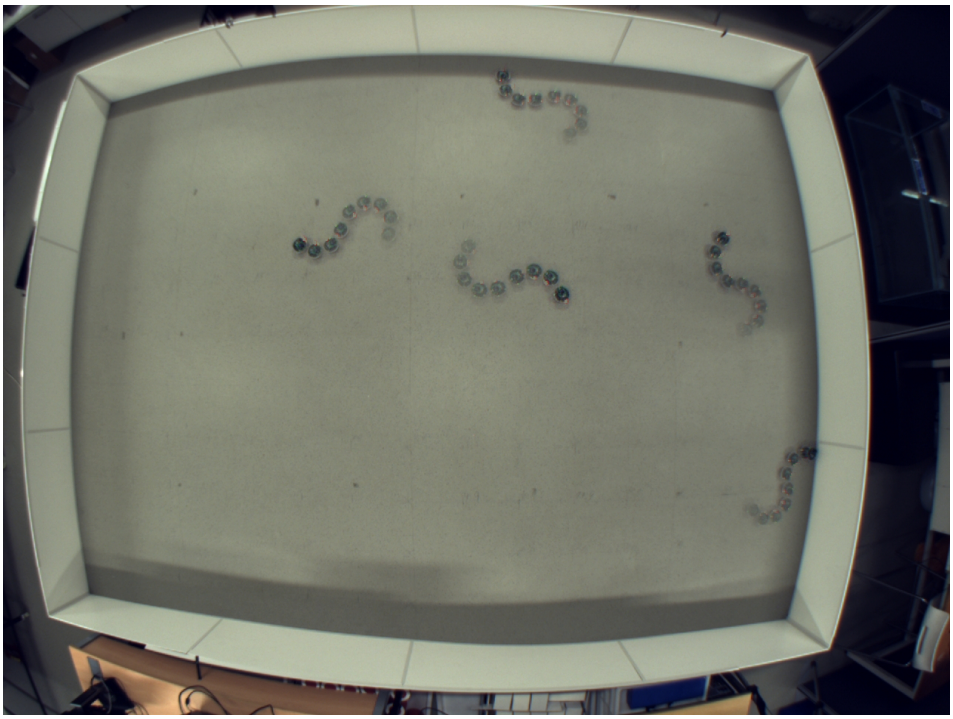
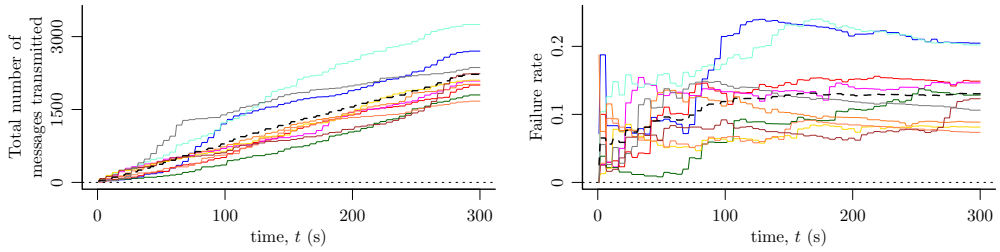


Figure 5.22: The sequence of 9 (superimposed) snapshots taken in a period of 3.2 s from one of the ten trials in which five e-pucks performs synchronous movements while avoiding collision. Snapshots are taken every 0.4 s.




(a) (b)

Figure 5.23: (a) The number of message transmission attempts by the router to the robots. (b) Failure rate.

# 6

# Probabilistic supervisory control of swarms of robots

In previous chapters, supervisory control theory (SCT) and associated design tools have been used to overcome some of the deficiencies of ad-hoc development. Given a formal description of the swarm's agents capabilities and their desired behaviour, the control source code was automatically generated. However, the traditional SCT is unable to formulate probabilistic controllers at the supervisor level. Probabilities, if needed, must be considered on the lowest level of the control structure, the operational procedures [22].

In this chapter, we extend the SCT framework to incorporate probabilistic generators. To formulate probabilistic controllers at the supervisor level, we propose a probabilistic SCT (pSCT) framework [149]. The framework combines the concept of probabilistic generators [39, 40] with support for marked states and the synthesis of monolithic [15], modular [16], and local modular supervisors [20, 19, 22]. The latter results in supervisors with more compact memory representation compared to the others.

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 [150], 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.

We recall that in the SCT's context, events can be uncontrollable or controllable. Uncontrollable events are related to the controller input, for example, from the sensors of a robot. Controllable events are 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 [18]. 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 illustrate and 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 [151]. pSCT automatically generates the controller's code and applies it on physical swarms of 25 and 100 Kilobot robots [38].

## 6.1 Choice problem

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.

During the synthesis of the supervisor the only concern is to avoid deadlocks and livelocks. Deadlocks and livelocks, in the context of generators, occur when the controller is trapped in a non-marked state that has no path to reach a marked state. Deadlocks imply that the system cannot evolve further, in contrast, livelocks imply that the controller can evolve further but only in a subset of states without reaching a marked state.

For example, let us consider the supervisor shown in Figure 6.1. 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.

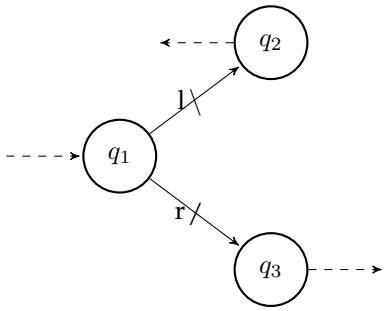



Figure 6.1: A choice problem between two enabled controllable events from state $q_1$.

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.[1]

The supervisor shown in Figure 6.2 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.

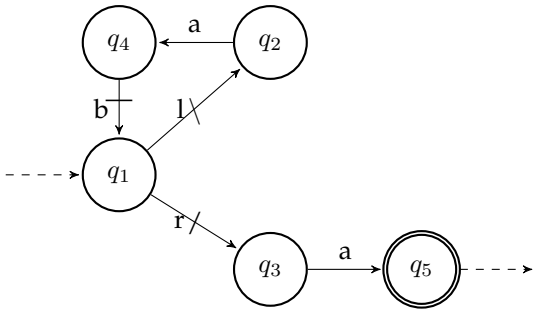



Figure 6.2: A choice problem with livelock. If the same transition ($l$) is repeatedly chosen from $q_1$ a livelock can occur. Some states of the supervisors are omitted.

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 6.3. 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 6.4). The uncontrollable

[1] If a controller is trapped in a single non-marked state this is called a deadlock. Deadlocks can be prevented during the synthesis.

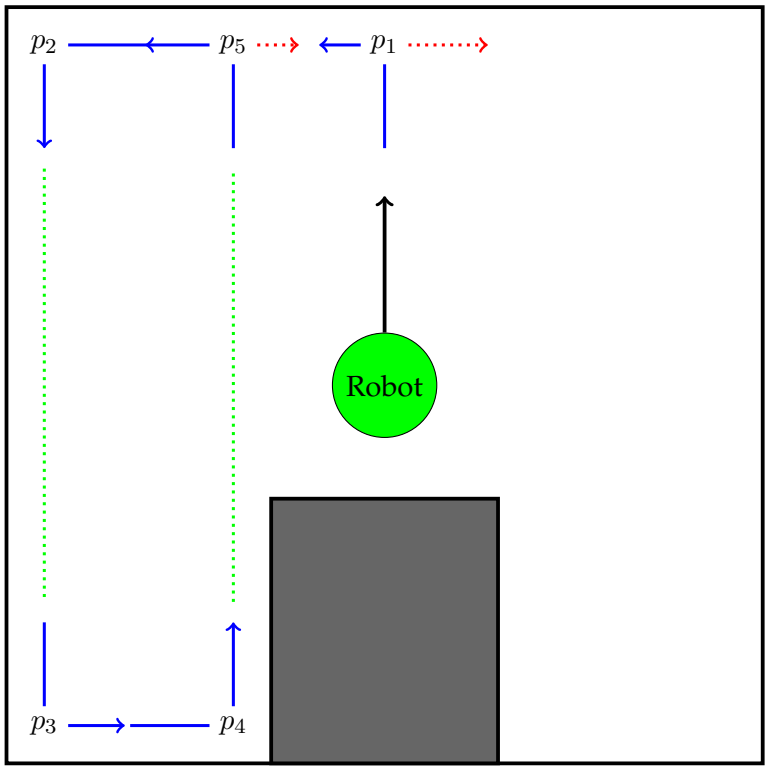



Figure 6.3: A task that under-performs due to the choice problem.

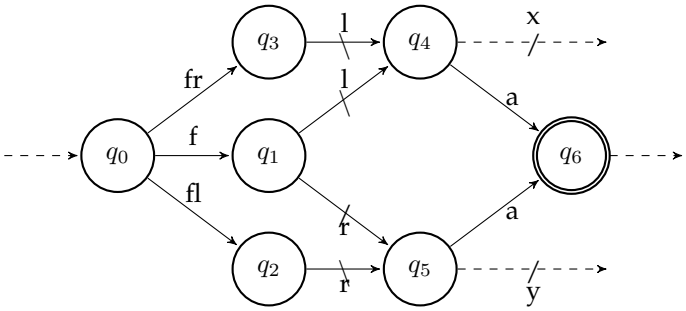



Figure 6.4: A choice problem without livelocks. Some states are omitted.

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 6.4). 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 6.4), 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.

Events could be chosen in particular sequence every time the state is achieved. This would avoid livelocks, as eventually all the controllable transitions will be chosen. However, it requires to keep track of the last selected controllable event of each state, increasing the amount of volatile memory used by the controller. Optionally, an equivalent supervisor with single controllable event per state could be computed. This computation will increase the size of the supervisor in program memory, which may not be desired or even feasible.

The alternation in sequence between the enabled controllable events can also lead to undesired implementation restriction, as shown in the examples in Figure 6.5. The use of any deterministic sequence of events in a particular state where the choice problem appears can also lead to undesired implementation restrictions. For example, let us assume that the deterministic sequence $l, r, l, \cdots$ is chosen, Figure 6.5 shows an example where an undesired implementation restriction appears. Note that for the configuration of this arena the robot is still not able to explore the entire arena, even though the supervisor does not impose this restriction. In this example, the considered robot move at the positions $p_1$, $p_5$, and $p_6$ lead the supervisor in Figure 6.4 to state $q_1$. In the position $p_1$ the controller chooses left. In the position $p_5$ the controller chooses right, when it is in position $p_6$ it chooses left again. Note that on the course from position $p_7$ to $p_2$ the robot does not trigger any collision when passing by $p_1$, as the collision only occurs if there is an obstacle in front of the robot. In the subsequent cycles the robot will always choose right in $p_5$ and left in $p_6$ leading to an undesired restriction.

Thus, a deterministic sequence can potentially lead to undesired implementation restrictions, in this case a limited exploration of the arena, assuming no noise being present. A better, and more common, approach is to choose one of the enabled controllable events randomly, as previously presented in Chapter 4. This will avoid live-

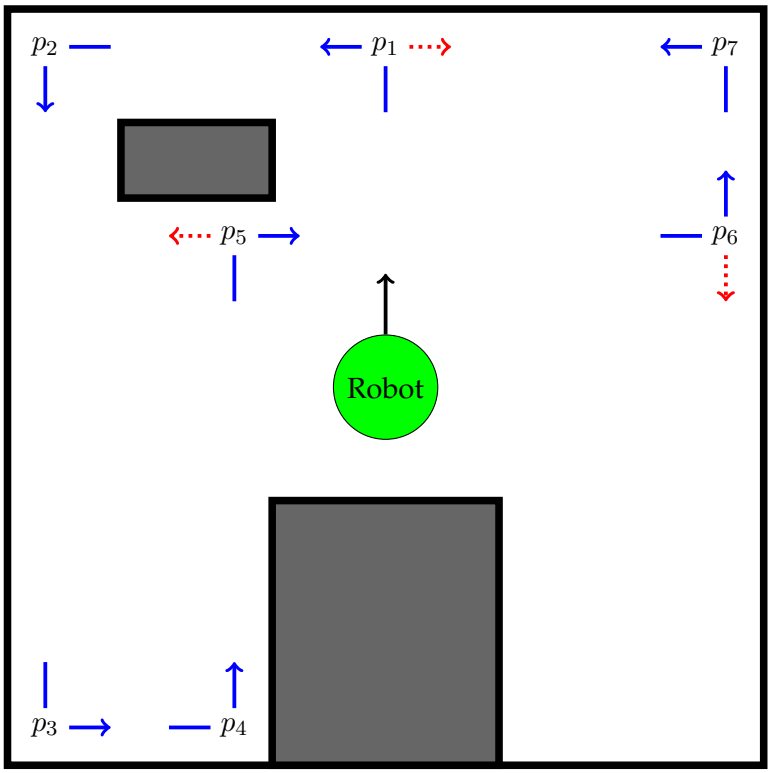



Figure 6.5: Example of a task that under-perform due to the choice problem.

locks and unspecified restrictions, as eventually all the controllable transitions will be chosen.

However, a uniformly random selection among the enabled controllable events is only a basic solution that cannot couple with different events' likelihood. 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 Chapters 3 and 4. In that case, the events move forward, turn left, and turn right occurred, statistically, in the same proportion. The differences were implemented in the operational procedures, where 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. Probabilistic generators are able to formally represent different likelihoods of controllable events.

## 6.2 Probabilistic generators

We recall that different definitions for probabilistic generators have been proposed. In [40] probabilistic generators are defined as:

$$G^p = \{Q, \Sigma, \delta, q_0, p\}, \tag{6.1}$$

where:

- $Q$ is a finite set of states.
- $\Sigma$ is a finite set of events, with $\Sigma = \Sigma_u \cup \Sigma_c$.
- $\delta : Q \times \Sigma \to Q$ is the partial transition function.
- $q_0$ is the initial state, where $q_0 \in Q$.

The probability of an event occurring in a particular state is:

$$p : Q \times \Sigma \to [0, 1], \tag{6.2}$$

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

$$\forall q \in Q, \sum_{e \in \Sigma} p(q, e) \leq 1. \tag{6.3}$$

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 [39], includes marked states, $Q_m$:

$$G^p = \{Q, \Sigma, \delta, q_0, Q_m, p\}. \tag{6.4}$$

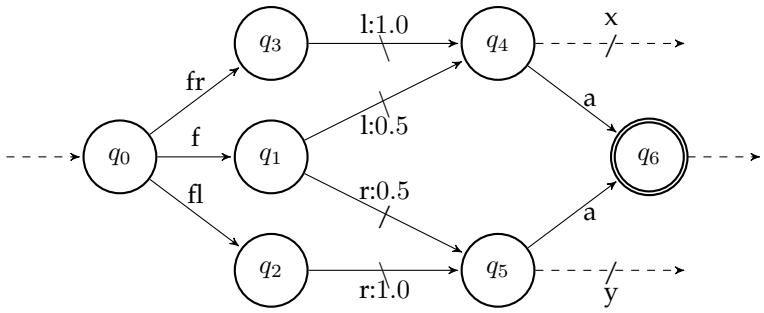



Figure 6.6: Example of a probabilistic generator derived from a non-probabilistic generator.

This definition has been applied to the synthesis of a supervisor defined by a single free behaviour model and a single specification [39]. The probabilistic supervisory control theory (pSCT) defines probabilistic generators as in Equation 2.34 but drops the restriction of Equation 6.3.

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 This allows the decomposition of the robots capabilities and the control specifications in several probabilistic generators; permitting the individual consideration of each component. In summary, $p(q, e)$ is the isolated probability of an event $e$ to occur in a state $q$, and $p'(q, e) = 1 - p(q, e)$ is the probability of the event $e$, in state $q$ do not occur. Non-probabilistic generators are a sub class of probabilistic generators where $p(q, e) \in \{0, 1\}$.

Non-probabilistic generators used by the traditional SCT can be expressed as probabilistic generators in our pSCT framework. Figure 6.6 shows the probabilistic version of the generator presented in Figure 6.4. The label of each transition is represented by $e : p$, where $e$ 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.

### 6.2.1 Operations for the synthesis of probabilistic supervisors

This section defines operations needed to synthesise supervisors based on the proposed probabilistic generators. The use of such operations will be detailed later using a case study.

#### 6.2.1.1 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:

$$p_n(q, e_{c_x}) = \begin{cases} p(q, e_{c_x}) \;/ \sum\limits_{e_c \in \Sigma_c} p(q, e_c) & \text{if } \sum\limits_{e_c \in \Sigma_c} p(q, e_c) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

Figure 6.7 shows an example of an PFG and its normalisation.

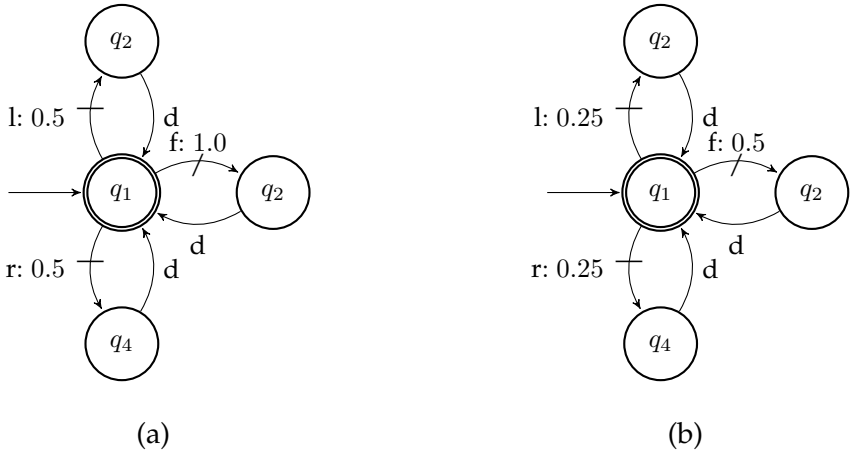



Figure 6.7: Example of the normalisation of the transitions probabilities.

#### 6.2.1.2 Synchronisation

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

$$G_a^p||G_b^p = (Q_a \times Q_b, \Sigma_a \cup \Sigma_b, \delta_{a||b}, (q_{0_a}, q_{0_b}), Q_{m_a} \times Q_{m_b}, p_{a||b}), \tag{6.6}$$

where

$$\delta_{a||b}((q_a, q_b), e) = \begin{cases} (\delta_a(q_a, e), \delta_b(q_b, e)) & \text{if } \delta_a(q_a, e)! \wedge \delta_b(q_b, e)! \\ (\delta_a(q_a, e), q_b) & \text{if } \delta_a(q_a, e)! \wedge e \notin \Sigma_b \\ (q_a, \delta_b(q_b, e)) & \text{if } \delta_b(q_b, e)! \wedge e \notin \Sigma_a \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{6.7}$$

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

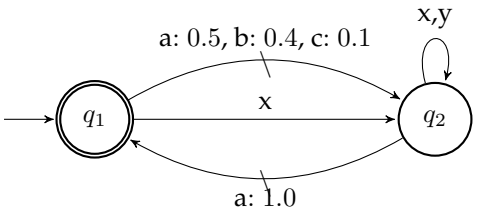
$$p_{a||b}((q_a, q_b), e_c) = \begin{cases} p_a(q_a, e_c) \times p_b(q_b, e_c) & \text{if } \delta_a(q_a, e_c)! \wedge \delta_b(q_b, e_c)! \\ p_a(q_a, e_c) & \text{if } \delta_a(q_a, e_c)! \wedge e_c \notin \Sigma_b \\ p_b(q_b, e_c) & \text{if } \delta_b(q_b, e_c)! \wedge e_c \notin \Sigma_a \\ 0 & \text{otherwise.} \end{cases} \tag{6.8}$$

Figure 6.8(c) shows an example of the synchronous composition of two probabilistic generators, $G_1^p$ (Figure 6.8(a)) and $G_2^p$ (Figure 6.8(b)). Note that the resulting generator (Figure 6.8(c)) is not necessarily normalised.

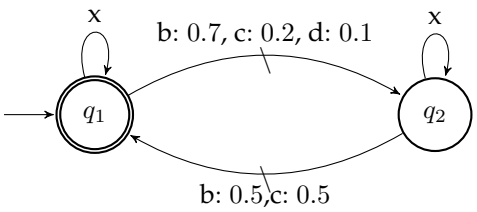
The associated probabilities for $G_1^p$ and $G_2^p$ gave in Figures 6.8(a) and 6.8(b) are given in Table 6.1.

Table 6.1: Associated probability for $G_1^p$ and $G_2^p$.

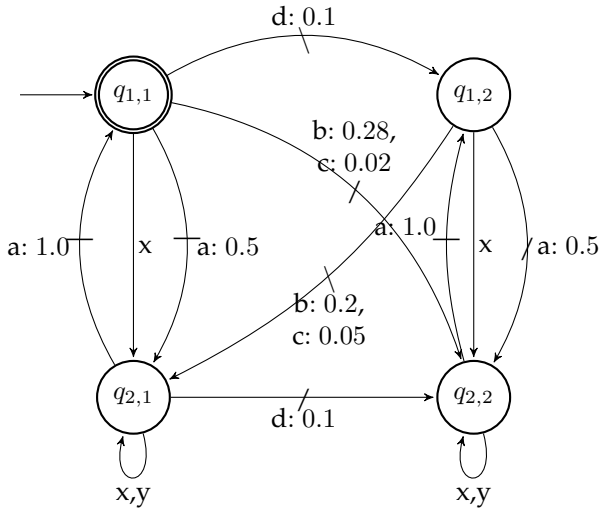
| State | $G_1^p$ | | | $G_2^p$ | | |
|---|---|---|---|---|---|---|
| | a | b | c | b | c | d |
| $q_1$ | 0.5 | 0.4 | 0.1 | 0.7 | 0.2 | 0.1 |
| $q_2$ | 1.0 | 0.0 | 0.0 | 0.5 | 0.5 | 0.0 |




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

Figure 6.8: Example of the synchronous composition of two probabilistic generators. (a) $G_1^p$ and (b) $G_2^p$, respectively are combined to form (c) $G^p_{(1,2)}$.

Equations 6.9 to 6.12 details the calculation of the probabilities resulting in the probabilistic generator $G^p_{1,2}$ shown in Figure 6.8(c).

$$\begin{array}{lcccccl}
p((q_1,q_1),a) & = & p_{g1}(q_1,a) & = & 0.5 & & \\
p((q_1,q_1),b) & = & p_{g1}(q_1,b) \times p_{g2}(q_1,b) & = & 0.4 \times 0.7 & = & 0.28 \\
p((q_1,q_1),c) & = & p_{g1}(q_1,c) \times p_{g2}(q_1,c) & = & 0.1 \times 0.2 & = & 0.02 \\
p((q_1,q_1),d) & = & p_{g2}(q_1,d) & = & 0.1 & &
\end{array} \tag{6.9}$$

$$\begin{array}{lcccccl}
p((q_1,q_2),a) & = & p_{g1}(q_1,a) & = & 0.5 & & \\
p((q_1,q_2),b) & = & p_{g1}(q_1,b) \times p_{g2}(q_2,b) & = & 0.4 \times 0.5 & = & 0.2 \\
p((q_1,q_2),c) & = & p_{g1}(q_1,c) \times p_{g2}(q_2,c) & = & 0.1 \times 0.5 & = & 0.05 \\
p((q_1,q_2),d) & = & p_{g2}(q_2,d) & = & 0.0 & &
\end{array} \tag{6.10}$$

$$\begin{array}{lcccccl}
p((q_2,q_1),a) & = & p_{g1}(q_2,a) & = & 1.0 & & \\
p((q_2,q_1),b) & = & p_{g1}(q_2,b) \times p_{g2}(q_1,b) & = & 0.0 \times 0.7 & = & 0.0 \\
p((q_2,q_1),c) & = & p_{g1}(q_2,c) \times p_{g2}(q_1,c) & = & 0.0 \times 0.2 & = & 0.0 \\
p((q_2,q_1),d) & = & p_{g2}(q_1,d) & = & 0.1 & &
\end{array} \tag{6.11}$$

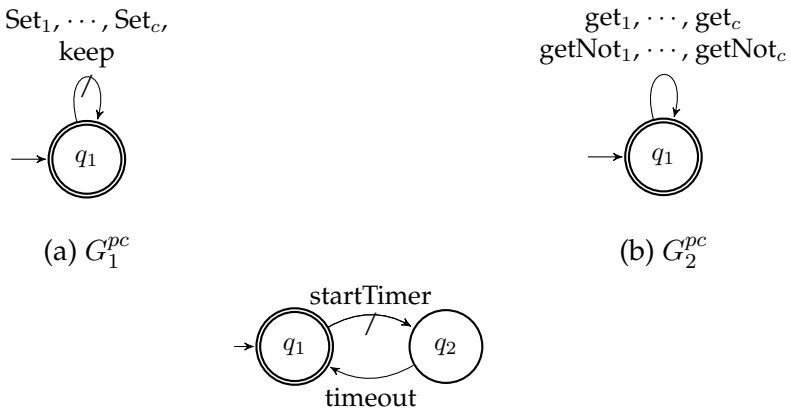
$$\begin{array}{lcccccc} p((q_2,q_2),a) & = & p_{g1}(q_2,a) & = & 1.0 & & \\ p((q_2,q_2),b) & = & p_{g1}(q_2,b) \times p_{g2}(q_2,b) & = & 0.0 \times 0.5 & = & 0.0 \\ p((q_2,q_2),c) & = & p_{g1}(q_2,c) \times p_{g2}(q_2,c) & = & 0.0 \times 0.5 & = & 0.0 \\ p((q_2,q_2),d) & = & p_{g2}(q_2,d) & = & 0.0 & & \end{array} \tag{6.12}$$

## 6.3 Graph colouring case study

Our case study derives from the classical graph colouring problem [151]. 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, where each robot has a unique local identifier considering its neighbourhood. Many swarm algorithms can benefit of a unique identification of each robot. However, unique hard coded values are as large as the swarm size, taking space of the communication message. Therefore, minimise the maximum identification value—the colour—and at the same time guarantee that no neighbour have the same identification number can be useful for many swarm robotics algorithms.

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 capabilities) for the graph colouring strategy are illustrated in Figure 6.9. Based on the availability of $c$ colours, free behaviour model $G_1^{pc}$ 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 colour selection. Free behaviour model $G_2^{pc}$ defines the uncontrollable events $get_x$ and $getNot_x$ with $x \in \{1, \cdots, c\}$. $get_x$ 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 $G_3^{pc}$ represents a timer that is triggered every 4 s.




(a) $G_1^{pc}$ (b) $G_2^{pc}$

(c) $G_3^{pc}$

Figure 6.9: Free behaviour models for the 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.

Table 6.2: Summary of events' definition for the graph colouring case study. In the controllability column controllable events are indicated by *C* and uncontrollable events are indicated by *U*.

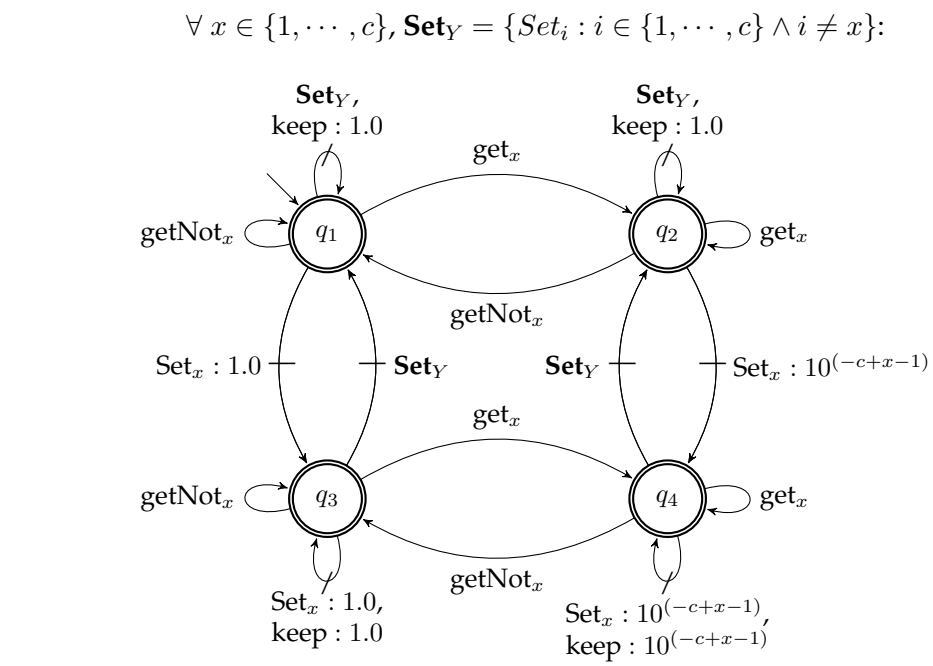
| event | controllability | definition |
|---|---|---|
| $Set_x$ | *C* | Selects colour $x \in \{1, \cdots, c\}$. |
| $keep$ | *C* | Preserves the previous selection. |
| $get_x$ | *U* | Neighbour has chosen the colour $x \in \{1, \cdots, c\}$. |
| $getNot_x$ | *U* | No neighbour has chosen the colour $x \in \{1, \cdots, c\}$. |
| $startTimer$ | *C* | Activates the timer. |
| $timeout$ | *U* | 2 s have elapsed since timer's activation. |

In Table 6.2 the events' definition is summarised.

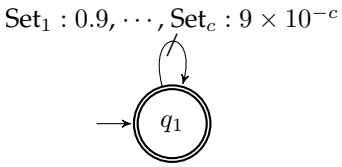
Figure 6.10 shows the control specification realising the graph colouring strategy. Specifications $E^{pc}_{(1,x)}$ (Figure 6.10(a)), with $x \in \{1, \cdots, c\}$, set a lower probability for selecting a colour $x$, if it is known that a neighbour already had selected it—in states $q_2$ and $q_4$. The 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 $E_2^{pc}$ (Figure 6.10(b)) defines the default probability of a colour being selected.[2] Colours have different probabilities, the $x$-th colour, with $x \in \{1, \cdots, c\}$, has

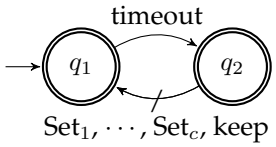
[2]Note that during synchronous composition, the probabilities defined in different specifications will get




(a) $E^{pc}_{(1,x)}$




(b) $E^{pc}_2$




(c) $E^{pc}_3$

Figure 6.10: Specifications for the 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.

probability $9 \times 10^{-x}$ 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 $E^{pc}_3$ (Figure 6.10(c)) establishes a waiting period for any change of colour to happen.

multiplied and normalised.

### 6.3.1 Supervisor synthesis

A monolithic probabilistic supervisor, $S^p$, using probabilistic generator is obtained by the synchronous composition of all free behaviour models and specifications into a single generator [16]. We will illustrate this process using the models for the graph colouring strategy shown in Figures 6.9 and 6.10. First, all free behaviour models are composed into a single generator:

$$G^{pc} = G_1^{pc} || G_2^{pc} || G_3^{pc}. \tag{6.13}$$

All specifications are composed into a single generator:

$$E^{pc} = E_{(1,1)}^{pc} || \ldots || E_{(1,c)}^{pc} || E_2^{pc} || E_3^{pc}. \tag{6.14}$$

E and G are composed together into a target language:

$$K^{pc} = G^{pc} || E^{pc}. \tag{6.15}$$

Finally, the monolithic probabilistic supervisor for the graph colouring problem $S^{pc}$ is the maximal controllable sub-language of $K^{pc}$. $S^{pc}$ 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 state is referred to as bad, as the uncontrollable event cannot be disabled by the supervisor. The supervisor is non-admissible if it contains bad states. 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:

$$S^{pc} = SupC(G^{pc}, K^{pc}). \tag{6.16}$$

For the modular approach the free behaviour model used, $G^{mod,pc}$, is the same of the monolithic approach (i.e. $G^{mod,pc} = G$).

Table 6.3: Events used by the specifications and free behaviour models for the graph colouring case study.

| | | $E_1$ | $E_2$ | $E_3$ |
|---|---|---|---|---|
| $G_1$ | $Set_x$ | ✓ | ✓ | ✓ |
| | $keep$ | ✓ | | ✓ |
| $G_2$ | $get_x$ | ✓ | | |
| | $getNot_x$ | ✓ | | |
| $G_3$ | $startTime$ | | | |
| | $timeout$ | | | ✓ |
| local models | | $G_1^{loc,pc}$ | $G_2^{loc,pc}$ | $G_3^{loc,pc}$ |

The target languages for the modular approach, $K_x^{mod,pc}$, are:

$$\begin{array}{lcl} K_{(1,i)}^{mod,pc} & = & E_{(1,i)}||G^{mod,pc} : \forall i \in \{1, \cdots, c\} \\ K_2^{mod,pc} & = & E_2||G^{mod,pc} \\ K_3^{mod,pc} & = & E_3||G^{mod,pc}. \end{array} \tag{6.17}$$

The modular supervisors are:

$$\begin{array}{lcl} S_{(1,i)}^{mod,pc} & = & SupC(G^{mod,pc}, K_{(1,i)}^{mod,pc}) \forall i \in \{1, \cdots, c\} \\ S_2^{mod,pc} & = & SupC(G^{mod,pc}, K_2^{mod,pc}) \\ S_3^{mod,pc} & = & SupC(G^{mod,pc}, K_3^{mod,pc}). \end{array} \tag{6.18}$$

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 [22]. 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 in the specification. Table 6.3 shows the relation of events for each specification for the graph colouring case study. The local free behaviour models, $G_x^{loc,pc}$, and the generators that realise the target language, $K_x^{loc,pc}$, are:

$$\begin{array}{lcl|lcl} G_1^{loc,pc} & = & G_1||G_2 & K_{(1,i)}^{loc,pc} & = & E_{(1,i)}||G_1^{loc,pc} : \forall i \in \{1, \cdots, c\} \\ G_2^{loc,pc} & = & G_1 & K_2^{loc,pc} & = & E_2||G_2^{loc,pc} \\ G_3^{loc,pc} & = & G_1||G_3 & K_3^{loc,pc} & = & E_3||G_3^{loc,pc}. \end{array} \tag{6.19}$$

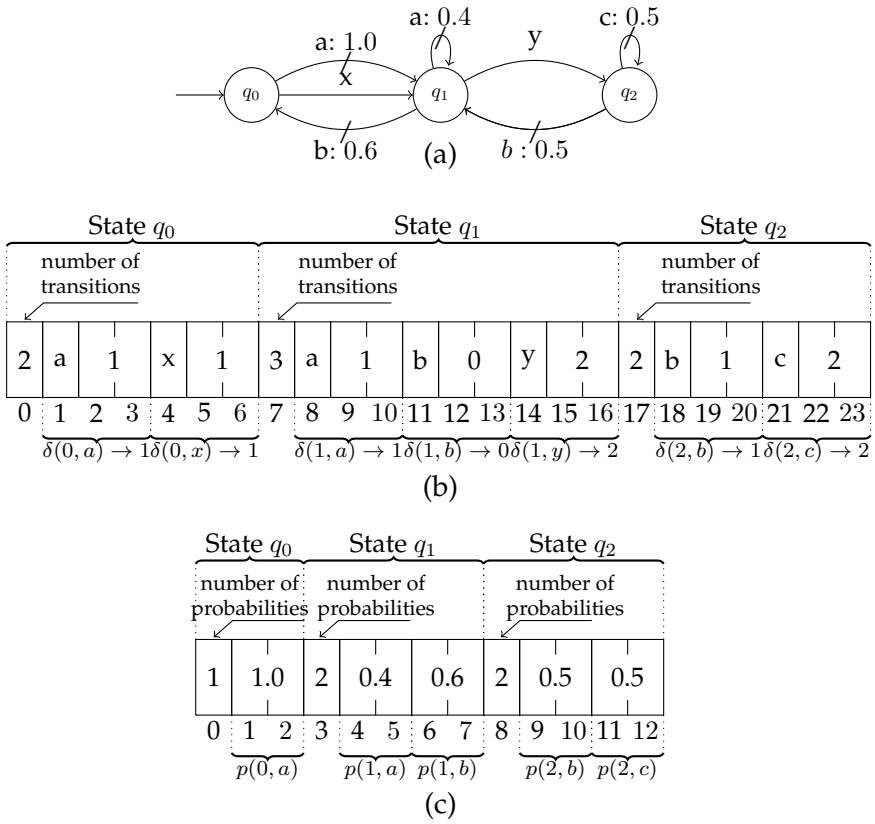



Figure 6.11: The memory representation of a probabilistic generator. (a) The probabilistic generator; (b) the partial transition function representation [1]. The first element of each state represents the number of outgoing transitions. It is followed by blocks of three elements, which detail the event that triggers the transition and the resulting state. (c) Representations of the probabilities of controllable transitions.

The local modular supervisors are:

$$\begin{aligned} S_{(1,i)}^{loc,pc} &= SupC(G_1^{loc,pc}, K_{(1,i)}^{loc,pc}) \forall i \in \{1, \cdots, c\} \\ S_2^{loc,pc} &= SupC(G_2^{loc,pc}, K_2^{loc,pc}) \\ S_3^{loc,pc} &= SupC(G_3^{loc,pc}, K_3^{loc,pc}). \end{aligned} \tag{6.20}$$

## 6.4 Implementation

The use of probabilistic deterministic finite generators under the pSCT framework requires two changes in the implementation, previously presented in [56]. 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.

### 6.4.1 Memory representation

Consider the probabilistic supervisor shown in Figure 6.11(a). Figures 6.11(b–c) illustrate the data structure that stores the supervisor in memory. In Figure 6.11(b) the representation of the partial transition function, $\delta$, is shown [1]. 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 6.11(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 ($\delta$) representation as each set is stored in the same order (see Figure 6.11(b)).

### 6.4.2 Probabilistic generator player

The probabilistic generator player (pGP) executes the generators realising the supervisors. We modify the traditional generator player presented in [56] 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 6.8 and it is normalised as defined

in Equation 6.5. The monolithic supervisor can be normalised at the time it is being synthesised (prior to run-time). However, the normalisation for local modular supervisor can only occur after all synchronisation operations are performed, as, in general:

$$Norm(S^{pc}) \quad = \quad Norm(S_1^{loc,pc}||\cdots||S_i^{loc,pc}) \quad \neq \quad Norm(S_1^{loc,pc})||\cdots||Norm(S_i^{loc,pc}). \tag{6.21}$$

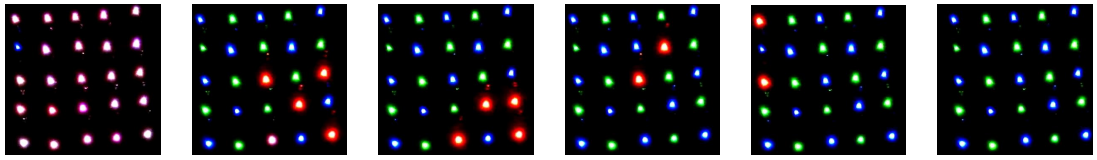
For that reason, the probabilistic generator player needs to incorporate the calculation of the normalised probabilities.

## 6.5 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.

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 $5 \times 5$ 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 $10 \times 10$ 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.



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

Figure 6.12: 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.

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. Table 6.4 details the results for all cases.

Table 6.4: Total number of states and transitions for each case study when using monolithic, modular, and local modular synthesis approaches, respectively. Data corresponds to the target language $K$ and supervisor $S$ using $c = 4$. The best results are highlighted in bold.

| | | Monolithic | | Modular | | Local modular | | |
|---|---|---|---|---|---|---|---|---|
| | | K | S | K | S | K | S | |
| graph colouring | states | 320 | | 240 | 38 | 37 | **21** | **20** |
| | transitions | 3600 | | 2480 | 521 | 507 | **226** | **220** |

Figure 6.12 shows snapshots taken from one of the experimental trials with 25 robots[3].

Figure 6.13 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 6.14(a) and the proportion of connections among neighbours with different colours is shown in Figure 6.14(b).

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

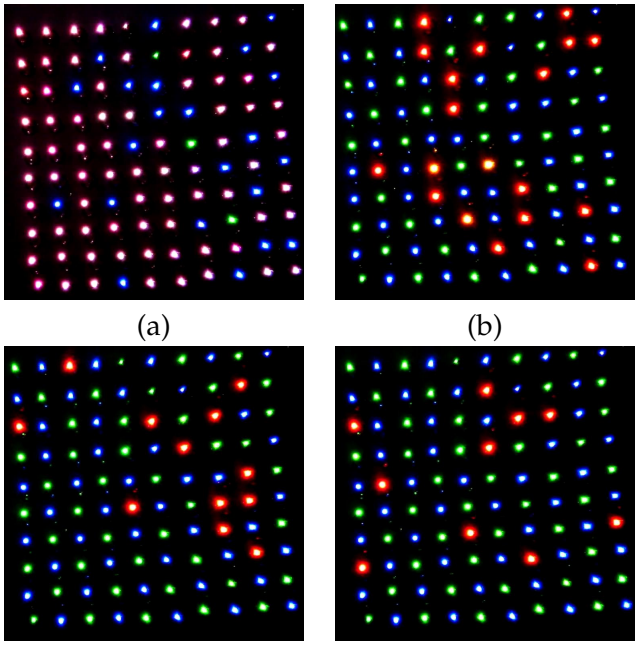



Figure 6.13: A sequence of snapshots of one of the two trials where 100 Kilobots performed the distributed graph colouring algorithm: Photos (a-d) show the experiment after 0 s, 900 s, 1800 s, and 2700 s.

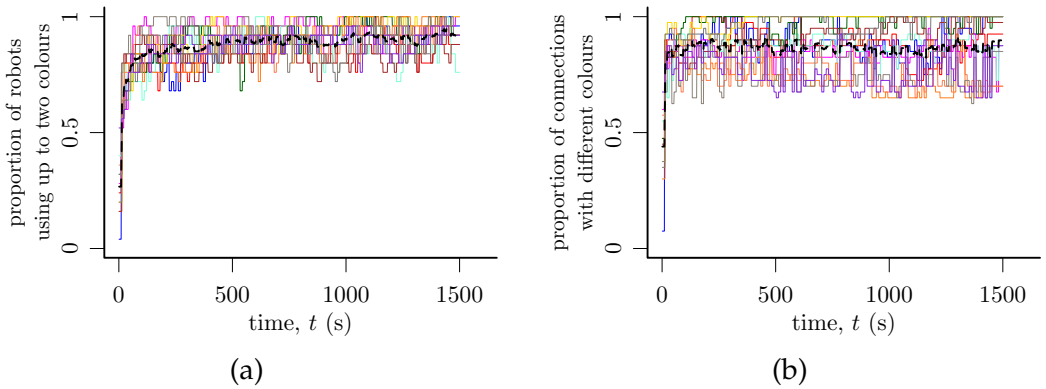
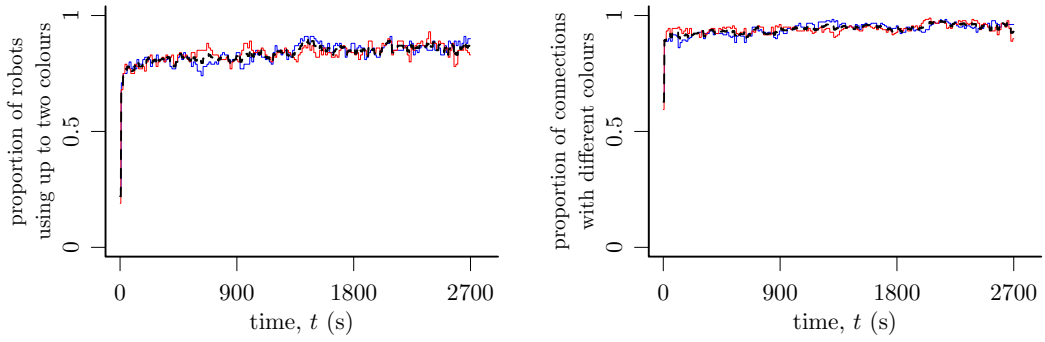



Figure 6.14: (a) The proportion of robots using up to 2 colours and (b) the proportion of the connections among neighbours with different colours in the 25 robots trials. Each coloured line represents one experimental trial. The thick black dashed line indicates the mean.

The proportion of robots using up to 2 colours for the 100 robots trials is shown in Figure 6.15(a) and the proportion of connections among neighbours with different colours is shown in Figure 6.15(b).




(a) (b)

Figure 6.15: (a) The proportion of robots using up to 2 colours and (b) the proportion of the connections among neighbours with different colours in the 100 robots trials.

Ideally, the robot's communication range would allow it to form connections only with their immediate neighbour 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 optimises both measured metrics simultaneously. The strategy succeeds in rapidly converging towards these solutions, though a small percentage of errors remain.

## 6.6 Segregation and group formation cases

In Sections 3.1.2 and 3.1.5 the segregation and group formation case studies were presented. Both cases were modelled using non-probabilistic generators. In these strategies, it was desired that the robots perform a random walk where they are moving forward (event moveFW) for more time than turning clockwise (event turnCW) or counter-clockwise (event turnCCW). The solution using the traditional SCT was to define such behaviour in the operational procedures, as shown in Table 6.5(a).

Table 6.5: Operational procedures reduction by the use of pSCT for the Kilobot platform for the segregation and group formation case studies.

```
int moveTimeout;
void callback_moveFW( void* data ){
    moveTimeout = 20;
    setMove( MOVE_FW );
}
void callback_turnCW( void* data ){
    moveTimeout = 10;
    setMove( TURN_CW );
}
void callback_turnCCW( void* data ){
    moveTimeout = 10;
    setMove( TURN_CCW );
}
```

(a)

```
void callback_moveFW( void* data ){
    setMoveFor10Sec( MOVE_FW );
}
void callback_turnCW( void* data ){
    setMoveFor10Sec( TURN_CW );
}
void callback_turnCCW( void* data ){
    setMoveFor10Sec( TURN_CCW );
}
```

(b)

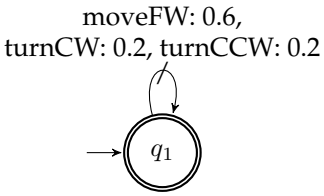



Figure 6.16: Specification defining a higher likelihood for the forward movement than for turning for the segregation and group formation case studies.

The probabilistic supervisory control theory approach enables us to define (and document) at the formal specification level the information that the forward movement is more prominent than turning clockwise or turning counter-clockwise (see Figure 6.16). This also leads to a simpler implementation of the operational procedures, as shown in Table 6.5(b).

## 6.7 Summary

In this chapter, we proposed pSCT, a probabilistic supervisory control theory (pSCT) framework. The framework uses a form of probabilistic generators to model probabilistic processes and, thereby, 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 searched 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 [152].

# 7

# Conclusion

This thesis explored the use of supervisory control theory (SCT) [15, 16, 17] to synthesise, in a formal manner, controllers for swarm robotics systems. It presented a framework that facilitates the development process from specification to implementation. The developers would define what the robots of the swarm could do (capabilities) and what they should do (specifications). In particular, they would provide, via a graphical user interface, formal languages, which model the capabilities and specifications. By combining the languages, SCT would then synthesise a new language, called supervisor, which controls each robot of the swarm. In particular, the supervisor would restrict a robot's set of possible actions to those that would not cause a violation of the specifications.

In this thesis, we used a type of state machines called generators to realise formal languages. Generators represent the class of regular languages. Generators differ from finite state automata in two aspects: (1) finite state automata recognise if a given word is part of the language, whereas generators produce these words; (2) finite state automata have a total transition function, whereas generators have a partial transition function. Note that SCT could be used with other classes of languages or be realised by other techniques as well. We opted for generators because state machines are commonly used to represent controllers of swarm robotics systems. However, most of these controllers have been implemented through ad-hoc development. The use of regular languages realised by generators in SCT is expected to enable a smooth transition from this ad-hoc development to more formal controller synthesis approaches. This, in turn, would facilitate the development of more reliable control software for swarm robotic systems, paving the way for such systems to be considered in real-world applications.

One advantage of the SCT framework is automatic code generation. In other words, both the control logic (represented by the supervisors) and source code (for the implementation of the control logic) can be automatically derived from the capabilities

and specifications provided by the developer. We demonstrated this for two robotic platforms and swarms of up to 600 physical robots. We also showed that the same supervisor could be executed on multiple platforms, as long as they offered the required capabilities.

This thesis builds on the works [20, 19, 22], in which a control structure and the local modular approach are introduced. The supervisory control theory was successfully applied considering three different aspects: deterministic discrete control over local events (Chapters 3 and 4), deterministic discrete control over global events (Chapter 5), and probabilistic discrete control over local events (Chapter 6).

In Chapter 3, we presented how to model the capabilities and specifications of a given system using five case studies. Three case studies found in literature and two novel strategies are used to didactically present how swarm robotics behaviours can be modelled using regular languages. Three methods for the supervisor synthesis are presented: monolithic, modular, and local modular.

In Chapter 4, the implementation of the supervisors obtained on Chapter 3 were discussed. We developed a software tool capable of automatically producing source code from the supervisors for two different swarm robotics platforms: Kilobot [38] and e-puck [37]. We validated our approach by conducting systematic experiments with up to 600 physical robots.

In Chapter 5, we showed how events occurring in different robots can be shared within the swarm. Robots in a swarm can be seen as isolated systems; work in a manufacturing context has shown how different systems can interact through the SCT framework. However, such systems operate in a hierarchical manner not compatible with swarm robotics principles. The proposed concept of global events allows such interactions without the need of a higher hierarchical level centralised controller, making it suitable for swarm robotics applications.

In Chapter 6, probabilistic generators were applied in the context of swarm robotics. It was shown that probabilistic generators within the probabilistic SCT (pSCT) framework offer an elegant solution to the choice problem. In fact, probabilistic generators can be seen as a super class of generators. We showed how the likelihood of controllable events can be formally defined and how this concept can be applied to specify and implement a swarm strategy.

Formal methods will contribute to making swarm robotics systems more reliable and may push their introduction in real-world applications, assisting humans in their day to day activities and helping people to improve the world around us. Future work could build upon this thesis by using other representations for formal languages, validating semantic properties, distributing the controller, and exploring multi-level hierarchical control.

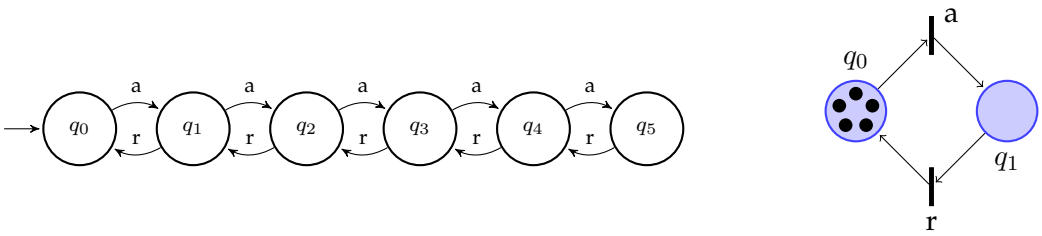
## 7.1 Future work

The work in this thesis can be seen as an attempt to establish the use of formal methods as standard practice in swarm robotics. Several questions remain unanswered at present: Will discrete event systems be sufficient for many real-world scenarios? How can macroscopic level models be used within the SCT? Will the proposed approach be suitable for more complex scenarios? What other extensions can be added to the SCT framework, and how they can meet swarm robotics needs? How can we tackle the need of ad-hoc code in the definition of operational procedures? How can automatic design techniques make use of the automatic code generation framework presented in this thesis? Future studies are therefore recommended.

### 7.1.1 Other representations for languages

State machines, such as generators, are a resource used by the swarm robotics community. This makes the use of generators a promising approach to advance the use of formal methods in the swarm robotics community. However, many other approaches exist and could be adequate for some applications.

Formal languages can be realised by Petri-nets. One advantage of Petri-nets is the possibility to represent some models in a more compact way when compared to generators. For example Figure 7.1(a) represents a generator that specifies the avoidance of overflow and underflow of a buffer of size $5$. Figure 7.1(b) shows the same specification using a Petri-net. The event $a$ is related to adding an item to the buffer and the event $r$ is related to removing an item from the buffer. Both formal machines represent the same regular language, but Petri-nets enable a more compact representation.




(a) (b)

Figure 7.1: Buffer specification to avoid underflow and overflow with a generator (a) and a Petri-net (b). Both formal machines represent the same regular language, but the Petri-net enables a more compact representation.

Petri-nets can be bounded or unbounded. In a bounded Petri-net all its places have up to a limited number of tokens. Bounded Petri-nets express the class of regular languages, as they can be converted to deterministic finite automata [119]. According to [119], bounded Petri-nets can be used within the SCT framework. In an unbounded Petri-net places may have an unlimited number of tokens. The computational power of unbounded Petri-nets is bigger than that of generators but strictly weaker than that of Turing machines [153]. Unbounded Petri-nets can have an infinite number of states.

The investigation of the properties in more powerful languages class, such as context-free languages (realised by push-down automata) or the languages realised by unbounded Petri-nets, could be of interest in many applications too. Future work could investigate how powerful language classes could model and control more complex swarm robotic systems.

### 7.1.2 Formal verification

This thesis was concerned with the validation of safety and non-blocking properties, which are system-theoretic or control-theoretic properties and are related only to the structure of the supervisor. According to [119], those properties include controllability, no conflict, no blocking and observability. In future works, techniques such as model checking could be applied to verify properties, defined by temporal logic specifications, of swarm robotics systems that use the SCT framework as presented in this thesis.

### 7.1.3 Transparent distribution of the controller

One issue to be overcome in swarm robotics is the limited computational resources on a single robot. To be able to implement large control algorithms, the controller could be distributed among the swarm, which provides more resources than a single robot. However, the distribution of the control comes at the cost of extra complexity in the control design. The open question is how the SCT could assist by providing a transparent distribution of control using formal methods as an alternative to the ad-hoc development of distributed control in swarm robotics. This would result in a more reliable controller that can be transparently distributed without additional design complexity.

A possible solution is to take advantage of the modularity of two of the three synthesis methods applied in this thesis: the modular approach and the local modular approach. The proposed idea consists of distributing $m$ local modular supervisors over a set of robots that can collaboratively calculate the next state and enable controllable events of each other. In the distributed approach, the group of $m$ modular or local modular supervisors $Z = \{S_i \mid \forall i \in [1, 2, \cdots, m]\}$ can be distributed over the group of $r$ robots $R = \{R_j \mid \forall j \in [1, 2, \cdots, r]\}$. Robot $R_j$ would store $Z_j \subseteq Z$ in such a way that each local modular supervisor would be stored in at least one robot, that is $Z = \bigcup_{j=1}^{r} Z_j$. Thus, each robot $R_j$ would keep a vector of its current state, $V_j^c$, for each supervisor in $Z$. The robot would perform a request to the others in two situations: (1) to determine the available controllable events for its current state vector, $V_l^c$, and (2) to determine its next state for each supervisor when an event $e$ occurs.

### 7.1.4 Multi-level hierarchical control

In the manufacturing context, the supervisory control theory has been applied from the coordination control of several devices to higher-level production planning systems. The lower level control of such devices is usually not approached by the SCT. The SCT acts by commanding the start of operations in the manufacturing plant—for example; a robot must load a milling cutter machine with a part that is located in a conveyor—without being concerned with how this is done.

In this thesis, on the other hand, we investigated how SCT can be used to synthesise these lower level controllers in the context of swarm robotics. We presented several cases that consist of a single behaviour dealing directly and in real time with sensors

and actuators of the robots. Each of these single behaviours can be seen as modules, which are analogous to machine operations in a manufacturing cell. SCT could be applied for the hierarchical controller to coordinate the activation of single behaviours in each robot of a swarm. The coordinator will be implemented in each robot's control structure without any centralised controller.

# References


[1] 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.

[2] J. Bishop, "Stochastic searching networks," in *Proceedings of 1st IEE Conference Artificial Neural Networks*, 1989, pp. 329–331.

[3] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[4] M. Dorigo and G. Di Caro, "New ideas in optimization," D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, Eds. Maidenhead, UK, England: McGraw-Hill Ltd., UK, 1999, ch. The Ant Colony Optimization Meta-heuristic, pp. 11–32.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[6] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Swarm intelligence: From natural to artificial systems," *Oxford University Press*, 1999.

[7] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella, "Evolving self-organizing behaviors for a swarm-bot," *Autonomous Robots*, vol. 17, no. 23, pp. 223–245, 2004.

[8] G. Beni, "From Swarm Intelligence to Swarm Robotics," in *Swarm Robotics*, 2005, pp. 1–9.

[9] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm Robotics*, ser. Lecture Notes in Computer Science, E. Şahin and W. M. Spears, Eds., vol. 3342. Springer, 2005, pp. 10–20.

[10] E. Şahin and A. Winfield, "Special issue on swarm robotics," *Swarm Intelligence*, vol. 2, no. 2, pp. 69–72, 2008.

[11] 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.

[12] W. Li, A. Miyazawa, P. Ribeiro, A. Cavalcanti, J. Woodcock, and J. Timmis, "From formalised state machines to implementations of robotic controllers," in *Proc. of the 2016 Int. Symposium on Distributed Autonomous Robotic Systems (DARS 2016)*, 2016.

[13] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys*, no. 4, pp. 626–643, 1996.

[14] S. C. Reghizzi, *Formal Languages and Compilation*, ser. Texts in Computer Science. Dordrecht: Springer, 2009.

[15] 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.

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

[17] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[18] 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.

[19] 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.

[20] ——, "Modular control of composed systems," in *Proceedings of the 2000 American Control Conference*. Piscataway, NJ: IEEE, 2000, pp. 4051–4055.

[21] J. Liu and H. Darabi, "Ladder logic implementation of Ramadge-Wonham supervisory controller," in *2002 IEEE 6th International Workshop on Discrete Event Systems*. Piscataway, NJ: IEEE, 2002, pp. 383–389.

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

[23] R. D. Barreta and C. R. Torrico, "Máquinas de mealy e moore para implementação de controle supervisório de sistemas a eventos discretos em microcontroladores," *Anais do XVII Congresso Brasileiro de Automática - CBA2008. (In Portuguese)*, 2008.

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

[25] Y. Silva and M. Queiroz, "Formal synthesis, simulation and automatic code generation of supervisory control for a manufacturing cell," *ABCM Symposium Series in Mechatronics - Vol. 4*, pp. 418–426, 2010.

[26] A. B. Leal, D. L. L. Cruz, and M. S. Hounsell, "PLC-based implementation of local modular supervisory control for manufacturing systems," in *Manufacturing System*, F. A. Aziz, Ed. Rijeka, Croatia: InTech, 2012, pp. 159–182.

[27] 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.

[28] C. Perrard and N. Andreff, "Control of a team of micro-robots for non-invasive medical applications," in *6th National Conference on Control Architectures of Robots*. Grenoble, France: INRIA Grenoble Rhône-Alpes, May 2011.

[29] S. Nolfi, G. Baldassarre, and D. Marocco, "The importance of viewing cognition as the result of emergent processes occurring at different time scales," in *Proceedings of the Third International Symposium on Human and Artificial Intelligent Systems - Dynamic Systems Approach for Embodiment and Sociality*, T. Asakura and K. Murase, Eds. Fukui, Japan: Fukui University, 2002, pp. 63–76.

[30] O. S. Erkin Bahceci and E. Sahin, "A review: Pattern formation and adaptation in multi-robot systems," Carnegie Mellon University, Tech. Rep. CMU-RI-TR-03-43, October 2003.

[31] A. Tsalatsanis, A. Yalcin, and K. Valavanis, "Optimized task allocation in cooperative robot teams," in *Proccedings of the 17th Mediterranean Conference on Control and Automation (MED'09)*. Piscataway, NJ: IEEE, 2009, pp. 270–275.

[32] A. Tsalatsanis, A. Yalcin, and K. P. Valavanis, "Dynamic task allocation in cooperative robot teams," *Robotica*, vol. 30, no. 5, pp. 721–730, 2012.

[33] R. Groß and M. Dorigo, "Group transport of an object to a target that only some group members may sense," in *Parallel Problem Solving from Nature – 8th International Conference (PPSN VIII)*, ser. Lecture Notes in Computer Science, vol. 3242. Springer Verlag, Berlin, Germany, 2004, pp. 852–861.

[34] J. Chen, M. Gauci, W. Li, A. Kolling, and R. Groß, "Occlusion-based cooperative transport with a swarm of miniature mobile robots," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 307–321, 2015.

[35] M. Dorigo, E. Tuci, V. Trianni, R. Groß, S. Nouyan, C. Ampatzis, T. H. Labella, R. O'Grady, M. Bonani, and F. Mondada, "SWARM-BOT: Design and implementation of colonies of self-assembling robots," in *Computational Intelligence: Principles and Practice*, G. Y. Yen and D. B. Fogel, Eds. IEEE Computational Intelligence Society, NY, 2006, pp. 103–135.

[36] J. C. Knight, C. L. DeJong, M. S. Gibble, and L. G. Nakano, "Why are formal methods not used more widely?" in *Fourth NASA Formal Methods workshop*. NASA, 1997, pp. 1–12.

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

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

[39] 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.

[40] 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.

[41] A. J. C. Sharkey, "The application of swarm intelligence to collective robots," in *Advances in Applied Artificial Intelligence*, J. Fulcher, Ed. Idea Group Publishing, 2006, pp. 157–185.

[42] Y. Mohan and S. Ponnambalam, "An extensive review of research in swarm robotics," *Nature & Biologically Inspired Computing*, pp. 140–145, 2009.

[43] Y. Liu and K. M. Passino, "Swarm Intelligence: Literature overview," 2000.

[44] L. Bayindir and E. Sahin, "A review of studies in swarm robotics," *Turkish Journal of Electrical Engineering*, vol. 15, no. 2, 2007.

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

[46] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[47] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology,Intelligence,and Technology*. Cambridge, MA, USA: MIT Press, 2000.

[48] M. L. Minsky, *Computation: Finite and Infinite Machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1967.

[49] M. Gauci, J. Chen, W. Li, T. J. Dodd, and R. Groß, "Clustering objects with robots that do not compute," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS '14)*. Richland, SC: IFAAMS, 2014, pp. 421–428.

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

[51] F. Bergenti and A. Poggi, "Exploiting uml in the design of multi-agent systems," in *Engineering Societies in the Agents World: First International Workshop*, A. Omicini, R. Tolksdorf, and F. Zambonelli, Eds., 2000, pp. 106–113.

[52] M. Gauci, J. Chen, T. J. Dodd, and R. Groß, "Evolving aggregation behaviors in multi-robot systems with binary sensors," in *Proc. of the 2012 Int. Symposium on Distributed Autonomous Robotic Systems (DARS 2012)*, ser. Springer Tracts in Advanced Robotics, vol. 104. Springer-Verlag, Berlin, Germany, 2014, pp. 355–367.

[53] M. Gauci, , J. Chen, W. Li, T. J. Dodd, and R. Groß, "Self-organised aggregation without computation," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1145–1161, 2014.

[54] J. D. Bjerknes and A. F. T. Winfield, "On fault tolerance and scalability of swarm robotic systems," in *Proc. of the 2013 Int. Symposium on Distributed Autonomous Robotic Systems (DARS 2013)*, A. Martinoli, F. Mondada, N. Correll, G. Mermoud, M. Egerstedt, A. M. Hsieh, E. L. Parker, and K. Støy, Eds., 2013, pp. 431–444.

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

[56] 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.

[57] O. Soysal, "Probabilistic aggregation strategies in swarm robotic systems," in *in Proc. of the IEEE Swarm Intelligence Symposium*, 2005, pp. 325–332.

[58] S. Nouyan, A. Campo, and M. Dorigo, "Path formation in a robot swarm," *Swarm Intelligence*, vol. 2, no. 1, pp. 1–23, 2008.

[59] T. H. Labella, M. Dorigo, and U. L. D. Bruxelles, "Division of labor in a group of robots inspired by ants foraging behavior," *ACM Transactions on Autonomous and Adaptive Systems*, pp. 4–25, 2006.

[60] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Rob. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[61] J. H. Reif and H. Wang, "Social potential fields: A distributed behavioral control for autonomous robots." *Robotics and Autonomous Systems*, vol. 27, no. 3, pp. 171–194, 1999.

[62] K. Hosokawa, I. Shimoyama, and H. Miura, "Dynamics of self-assembling systems: Analogy with chemical kinetics," *Artificial Life*, vol. 1, no. 4, pp. 413–427, 1994.

[63] A. Martinoli, A. J. Ispeert, and L. Gambardella, "A probabilistic model for understanding and comparing collective aggregation mechanisms," in *Advances in Artificial Life (ECAL 99)*, ser. Lecture Notes in Artificial Intelligence 1674, D. Floreano, J.-D. Nicoud, and F. Mondada, Eds. Springer-Verlag, 1999, pp. 575–584.

[64] K. Lerman, A. Galstyan, A. Martinoli, and A. Ijspeert, "A macroscopic analytical model of collaboration in distributed robotic systems," *Artificial Life Journal*, vol. 7, no. 4, pp. 375–393, 2001.

[65] A. Martinoli, K. Easton, and W. Agassounon, "Modeling swarm robotic systems: A case study in collaborative distributed manipulation," *The International Journal of Robotics Research*, vol. 23, no. 4-5, pp. 415–436, 2004.

[66] K. Lerman and A. Galstyan, "Mathematical model of foraging in a group of robots: Effect of interference," *Auton. Robots*, vol. 13, no. 2, pp. 127–141, 2002.

[67] V. Trianni, T. Labella, R. Groß, E. Şahin, M. Dorigo, and J. Deneubourg, "Modeling pattern formation in a swarm of self-assembling robots," *Techinical Report TR/IRIDIA/2002-12*, 2002.

[68] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, pp. 226–234, 1997.

[69] L. Iocchi, D. Nardi, and M. Salerno, "Reactivity and deliberation: A survey on multi-robot systems," in *Balancing Reactivity and Social Deliberation in Multi-Agent Systems, From RoboCup to Real-World Applications (selected papers from the ECAI 2000 Workshop and additional contributions)*. London, UK, UK: Springer-Verlag, 2001, pp. 9–34.

[70] L. E. Parker, "Designing control laws for cooperative agent teams," in *In IEEE International Conference on Robotics and Automation*, 1993, pp. 582–587.

[71] J. Rushby, "Formal methods and digital systems validation for airborne systems," SRI International, Melon Park, CA, Tech. Rep., 1993.

[72] M. Massink, M. Brambilla, D. Latella, M. Dorigo, and M. Birattari, "On the use of bio-pepa for modelling and analysing collective behaviours in swarm robotics," *Swarm Intelligence*, vol. 7, no. 2–3, pp. 201–228, 2013.

[73] 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.

[74] A. Franci, V. Srivastava, and N. Ehrich Leonard, "A realization theory for bio-inspired collective decision-making," *ArXiv e-prints*, 2015.

[75] M. Brambilla, C. Pinciroli, M. Birattari, and M. Dorigo, "Property-driven design for swarm robotics," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, 2012, pp. 139–146.

[76] 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.

[77] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, V. Trianni, and M. Birattari, "An experiment in automatic design of robot swarms: Automode-vanilla, evostick, and human experts," in *Swarm Intelligence, ANTS 2014*, ser. LNCS, M. Dorigo, et al., Ed., vol. 8667. Berlin, Germany: Springer, 2014, pp. 25–37.

[78] 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.

[79] G. Francesca and M. Birattari, "Automatic design of robot swarms: Achievements and challenges," *Frontiers in Robotics and AI*, vol. 3, pp. 1–9, 2016.

[80] Y. Khaluf, M. Pace, F. Rammig, and M. Dorigo, "Integrals of markov processes with application to swarm robotics modelling," IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2012-020, December 2012.

[81] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, Universität Hamburg, 1962.

[82] J. King, R. Pretty, and R. Gosine, "Coordinated execution of tasks in a multiagent environment," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 33, no. 5, pp. 615 – 619, 2003.

[83] H. Costelha and P. Lima, "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. Richland, SC: IFAAMS, 2008, pp. 1187–1190.

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

[85] A. Giua, "Petri nets as discrete event models for supervisory control," Ph.D. dissertation, Rensselaer Polytechnic Institute, 1992.

[86] B. Lacerda and P. U. Lima, "On the notion of uncontrollable marking in supervisory control of petri nets," *IEEE Transactions on Automatic Control*, vol. 59, no. 11, pp. 3069 – 3074, 2014.

[87] E. Emerson, "Temporal and Modal Logic," in *Handbook of Theoretical Computer Science In van Leeuwen, J., (Ed.)*. Elsevier, 1990, pp. 996–1072.

[88] A. F. T. Winfield, J. Sa, M.-C. Fernández-Gago, C. Dixon, and M. Fisher, "On formal specification of emergent behaviours in swarm robotic systems," *International Journal of Advanced Robotic Systems*, vol. 2, no. 4, pp. 363–370, 2005.

[89] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas, "Symbolic planning and control of robot motion [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.

[90] C. Dixon, A. Winfield, and M. Fisher, "Towards temporal verification of emergent behaviours in swarm robotic systems," in *Towards Autonomous Robotic Systems*, ser. Lecture Notes in Computer Science, R. Groß, L. Alboul, C. Melhuish, M. Witkowski, T. Prescott, and J. Penders, Eds., vol. 6856. Berlin, Germany: Springer, 2011, pp. 336–347.

[91] 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.

[92] P. Gainer, C. Dixon, and U. Hustadt, *Probabilistic Model Checking of Ant-Based Positionless Swarming*. Springer International Publishing, 2016, pp. 127–138.

[93] S. Hauert, L. Winkler, J.-C. Zufferey, and D. Floreano, "Ant-based swarming with positionless micro air vehicles for communication relay," *Swarm Intelligence*, vol. 2, no. 2, pp. 167–188, 2008.

[94] D. L. Milutinovic and P. U. Lima, *Cells and Robots: Modeling and Control of Large-Size Agent Populations*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[95] C. Tomlin, G. Pappas, J. Kosecka, J. Lygeros, and S. Sastry, "Advanced air traffic automation: A case study in distributed decentralized control," in *Proceedings of the Workshop Control Problems in Robotics and Automation*. Berlin, Germany: Springer-Verlag, 1998, pp. 261–295.

[96] R. Fierro, A. Das, V. Kumar, and J. Ostrowski, "Hybrid control of formations of robots," in *Proceedings of ICRA 2001, IEEE International Conference on Robotics and Automation*. Piscataway, NJ: IEEE, 2001, pp. 157–162.

[97] J. M. McNew and E. Klavins, "Locally interacting hybrid systems with embedded graph grammars," in *2006 45th IEEE Conference on Decision and Control*. Piscataway, NJ: IEEE, 2006, pp. 6080–6087.

[98] J. M. McNew, E. Klavins, and M. Egerstedt, "Solving coverage problems with embedded graph grammars," in *Hybrid Systems: Computation and Control*, ser. LNCS, A. Bemporad, et al., Ed., vol. 4416. Berlin, Germany: Springer, 2007, pp. 413–427.

[99] D. Milutinovic and P. U. Lima, "Modeling and optimal centralized control of a large-size robotic population," *IEEE Trans. Robotics*, vol. 22, no. 6, pp. 1280–1285, 2006.

[100] M. Zavlanos, H. Tanner, A. Jadbabaie, and G. Pappas, "Hybrid control for connectivity preserving flocking," *IEEE Transactions on Automatic Control*, vol. 54, no. 12, pp. 2869–2875, 2009.

[101] A. Mesquita, "Exploiting stochasticity in multi-agent systems," Ph.D. dissertation, University of California, Santa Barbara, CA, 2010.

[102] A. R. Mesquita and J. P. Hespanha, "Jump control of probability densities with applications to autonomous vehicle motion," *IEEE Transactions on Automatic Control*, vol. 57, no. 10, pp. 2588–2598, 2012.

[103] H. Ehrig, "Introduction to the algebraic theory of graph grammars (a survey)," in *Proceedings of the International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*. London, UK, UK: Springer-Verlag, 1979, pp. 1–69.

[104] H. O. Fattorini, *Infinite dimensional optimization and control theory*, ser. Encyclopedia of mathematics and its applications ;. New York :: Cambridge University Press,, 1999.

[105] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[106] M. Gauci, R. Nagpal, and michael rubenstein, "Programmable self-disassembly for shape formation in large-scale robot collectives," in *Proc. of the 2016 Int. Symposium on Distributed Autonomous Robotic Systems (DARS 2016)*, 2016.

[107] J. Chen, M. Gauci, M. Price, and R. Groß, "Segregation in swarms of e-puck robots based on the brazil nut effect," in *Proc. of the 11th Int. Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2012*, 2012.

[108] J. Chen, M. Gauci, and R. Groß, "A strategy for transporting tall objects with a swarm of miniature mobile robots," in *Proc. of the 2013 IEEE Int. Conf. on Robotics and Automation, ICRA 2013*, 2013.

[109] F. Inácio, D. Macharet, and L. Chaimowicz, "United we move: Decentralized segregated robotic swarm navigation," in *Proc. of the 2016 Int. Symposium on Distributed Autonomous Robotic Systems (DARS 2016)*, 2016.

[110] S. Magnenat and Laboratory of Intelligent Systems, EPFL, Lausanne, "Enki reference documentation," Tech. Rep., 2005. [Online]. Available: http://lis2.epfl.ch/resources/download/doc1.0/libenki/

[111] F. Perez-Diaz, R. Zillmer, and R. Groß, "Firefly-inspired synchronization in swarms of mobile agents," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '15. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 279–286.

[112] N. Salomons, G. Kapellmann-Zafra, and R. Groß, "Human management of a robotic swarm," in *Proc. 17th Annual Conference Towards Autonomous Robotic Systems (TAROS 2016)*, ser. Lecture Notes in Artificial Intelligence, vol. 9716. Springer-Verlag, 2016.

[113] G. Kapellmann-Zafra, N. Salomons, A. Kolling, and R. Groß, "Human-robot swarm interaction with limited situational awareness," in *International Conference on Swarm Intelligence*. Springer, 2016, pp. 125–136.

[114] G. Kapellmann-Zafra, J. Chen, and R. Groß, "Using google glass in human–robot swarm interaction," in *Proc. 17th Annual Conference Towards Autonomous Robotic Systems (TAROS 2016)*, ser. Lecture Notes in Artificial Intelligence, vol. 9716. Springer-Verlag, 2016.

[115] A. Gutiérrez, E. Tuci, and A. Campo, "Evolution of neuro-controllers for robots alignment using local communication," *International Journal of Advanced Robotic Systems*, vol. 6, no. 1, pp. 25–34, 2009.

[116] A. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena, "Open e-puck range & bearing miniaturized board for local communication in swarm robotics," in *IEEE International Conference on Robotics and Automation, ICRA 2009.*, 2009, pp. 3111–3116.

[117] S. Trenkwalder, Y. Lopes, K. A., A. Christensen, R. Prodan, and R. Groß, "Openswarm: An event-driven embedded operating system for miniature robots," in *International Conference on Intelligent Robots and Systems, IROS 2016*, 2016, pp. 0–7.

[118] S. Y. Yan, *An introduction to formal languages and machine computation*. World Scientific Publishing Co. Pte. Ltd., 1998.

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

[120] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, 1956.

[121] ——, "On certain formal properties of grammars," *Information and Control*, vol. 2, no. 2, pp. 137–167, 1959.

[122] M. Sipser, *Introduction to the theory of computation: second edition*, 2nd ed. Boston: PWS Pub., 2006.

[123] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines - part i," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1013–1025, July 2005.

[124] A. Paz, *Introduction o Probabilistic Automata*. New York: Academic Press, 1971.

[125] R. C. Carrasco and J. Oncina, *Learning stochastic regular grammars by means of a state merging method*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 139–152.

[126] H. Ney, *Stochastic Grammars and Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 319–344.

[127] W.-G. Tzeng, "A polynomial-time algorithm for the equivalence of probabilistic automata," *SIAM J. Comput.*, vol. 21, no. 2, pp. 216–227, 1992.

[128] A. L. N. Fred, *Computation of Substring Probabilities in Stochastic Grammars*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 103–114.

[129] M. Young-Lai and F. W. Tompa, "Stochastic grammatical inference of text database structure," *Machine Learning*, vol. 40, no. 2, pp. 111–137, 2000.

[130] A. Cowley and C. Taylor, "Orchestrating concurrency in robot swarms," in *Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems*. Piscataway, NJ: IEEE, 2007, pp. 945–950.

[131] D. Gordon-Spears and K. Kiriakidis, "Reconfigurable robot teams: modeling and supervisory control," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 5, pp. 763–769, 2004.

[132] D. Silva, E. Santos, A. Vieira, and M. de Paula, "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*. Piscataway, NJ: IEEE, 2008, pp. 751–758.

[133] D. Mass, A. Pinotti, and A. Leal, "Síntese e implementação de controle supervisório monolítico para um ice maker," in *Anais do XIX Congresso Brasileiro de Automática, CBA*, vol. 19, 2012, pp. 5294–5301.

[134] S. Forschelen, J. van de Mortel-Fronczak, R. Su, and J. Rooda, "Application of supervisory control theory to theme park vehicles," *Discrete Event Dynamic Systems*, vol. 22, no. 4, pp. 511–540, 2012.

[135] J. Brzozowski, "Canonical regular expressions and minimal state graphs for definite events," *Mathematical Theory of Automata*, vol. 12, pp. 529–561, 1962.

[136] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems," in *2006 IEEE 8th International Workshop on Discrete Event Systems*. Piscataway, NJ: IEEE, 2006, pp. 384–385.

[137] C. Reiser, A. E. C. Cunha, and J. Cury, "The environment Grail for supervisory control of discrete event systems," in *2006 IEEE 8th International Workshop on Discrete Event Systems*. Piscataway, NJ: IEEE, 2006, pp. 390–391.

[138] K. Rudie, "The integrated discrete-event systems tool," in *2006 IEEE 8th International Workshop on Discrete Event Systems*. Piscataway, NJ: IEEE, 2006, pp. 394–395.

[139] L. Feng and W. Wonham, "TCT: A computation tool for supervisory control synthesis," in *2006 IEEE 8th International Workshop on Discrete Event Systems*. Piscataway, NJ: IEEE, 2006, pp. 388–389.

[140] Y. K. Lopes, "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), 2012.

[141] ——, "Online supplementary material." [Online]. Available: http://naturalrobotics.group.shef.ac.uk/supp/ykaszubowskilopes_thesis/

[142] R. Sreenivas, "On a weaker notion of controllability of a language k with respect to a language l," *IEEE Transactions on Automatic Control*, vol. 38, no. 9, pp. 1446–1447, 1993.

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

[144] R. J. Leduc, M. Lawford, and P. Dai, "Hierarchical interface-based supervisory control of a flexible manufacturing system," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 4, pp. 654–668, 2006.

[145] Y. K. Lopes, R. S. U. Rosso, A. B. Leal, E. Harbs, and M. d. S. Hounsell, "Finite automata as an information model for MES and supervisory control integration," in *Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing*, vol. 14, no. 1, 2012, pp. F–488–F–493.

[146] Y. K. Lopes, E. Harbs, A. B. Leal, and R. S. U. Rosso, "Proposta de implementação de controle supervisório em microcontroladores," in *Proceedings of the 10th Simpósio Brasileiro de Automação Inteligente (SBAI 2011) (In portuguese)*, 2011.

[147] "Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. - part 15.1: Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpans)," *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, pp. 0–580, 2005.

[148] H. Zimmermann, "Osi reference model–the iso model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425–432, Apr 1980.

[149] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Probabilistic supervisory control theory (psct) applied to swarm robotics," in *Proceedings of the 2017 International Conference on Autonomous Agents and Multiagent Systems (to appear)*, ser. AAMAS '17, 2017.

[150] 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.

[151] 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.

[152] 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.

[153] H.-C. Yen, "Introduction to petri net theory," in *Recent Advances in Formal Languages and Applications*, ser. Studies in Computational Intelligence, Z. Esik, C. Martin-Vide, and V. Mitrana, Eds. Springer, 2006, vol. 25, pp. 343–373.