UNIVERSITÉ LIBRE DE BRUXELLES
Ecole Polytechnique de Bruxelles
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Formal methods for the design and analysis of robot swarms

Manuele BRAMBILLA

Promoteur de Thèse:
Prof. **Marco DORIGO**

Co-Promoteur de Thèse:
Prof. **Mauro BIRATTARI**

Thèse présentée en vue de l'obtention du titre de
Docteur en Sciences de l'Ingénieur

Année académique 2013-2014

Abstract

In this dissertation, we tackle two of the main open problems in swarm robotics: design and verification. We do so by using formal methods, in particular model checking.

Designing and developing individual-level behaviors to obtain a desired swarm-level goal is, in general, very difficult, as it is difficult to predict and thus design the non-linear interactions of tens or hundreds of individual robots that result in the desired collective behavior. The most common approach to the design of robot swarms is the bottom-up approach: in a trial-and-error process, the designer develops and improves the behavior of the individual robots until the desired collective behavior is obtained. This approach is not optimal: being a non-structured approach, the quality of the obtained swarm depends completely on the ingenuity and expertise of the designer. Moreover, the trial-and-error process is time consuming and cannot provide guarantees on the obtained results in terms of system correctness and properties. In this dissertation, we present our novel contribution to the top-down design of robot swarms: property-driven design. Property-driven design is based on the concepts of prescriptive modeling and model checking. Using property-driven design it is possible to design robot swarms in a systematic way, realizing systems that are "correct by design". We demonstrate property-driven design on two case-studies: aggregation and foraging.

Developing techniques to *analyze* and verify a robot swarm is a necessary step in order to employ swarm robotics in real-world applications. Swarm robotics systems are particularly challenging to analyze: robot swarms can be observed and thus analyzed at the macroscopic or microscopic level. Usually, robot swarms are analyzed using computer simulations or rate equations. Computer simulations are an irreplaceable tool for analyzing the behavior of a robot swarm. However, they can only validate a small subset of the possible execution scenarios and are impractical to exhaustively verify a collective behavior. Rate equations, instead, are able to analyze the steady-state behavior of a robot swarm, that is, its long-term average behavior. However, with rate equations it is difficult to analyze the evolution of the swarm's behavior or any far-from-equilibrium anomalies. In this dissertation, we explore the use of formal methods, in particular of model checking, to analyze and verify the properties of robot swarms. Model checking allows us to formally describe a set of desired properties of a system, in a more powerful and precise way compared to other mathematical approaches, and verify whether a given model of a system satisfies them. We propose two different approaches: we develop a model of a robot swarm performing collective decision using Bio-PEPA, which allows researchers to define a single high level model that can then be analyzed using stochastic simulations, rate equations and model checking; we also develop a model of a collective transport behavior using KLAIM, which allows researchers to capture both the hardware and the behavior of the robots used.

Acknowledgments

I joined IRIDIA in 2008, when I was just a master student, eager to discover the world of swarm robotics. At that time, I did not know that IRIDIA is much more than a laboratory, IRIDIA is a family. During my time at IRIDIA many people have come, some have left, but all of them have contributed to create the fantastic atmosphere of IRIDIA and they will always have a special place in my heart.

First of all, I would like to thank Marco. He gave me the opportunity to join IRIDIA and has supported me during my years here. His "sink or swim" approach to supervision is a bit scary sometimes, but it gives you the necessary push to become a good researcher—if you don't sink :)

I would also like to thank Mauro. Mauro is not only a good friend, but an excellent supervisor and an amazing teacher. He has taught me so much during our everyday discussions and long meetings that it is difficult for me to thank him properly on paper.

A special thank to Carlo. Carlo has been there for me from the first day I joined IRIDIA (he arrived late, but still). He has been a great supervisor for my master thesis, and a great friend and colleague ever since. Carlo was one of the main reasons for me to join IRIDIA.

Another special mention for Eliseo, who is a great friend and has been the partner for countless discussions, scientific and not. I will always remember the many many hours spent working together on the review, for Swarmanoid, playing games and making pizza.

Thanks also to Gio. Giovanni is the perfect mixture of a very professional colleague, a great beer buddy and a psychopath. I will also always remember the time spent with Gio, both at IRIDIA, for a long time in the same office, and in "the place with the lake" together with our friend Faislossli.

Special thanks to Ale, a great friend, officemate and the best at complaining; thanks to Nithin, the always positive officemate.

Thanks also Gianpiero, for all the great work done together.

I want to thank all the damned souls of the Swarmanoid project I haven't mentioned yet: Ali, my Turkish brother; Arne, the brutal side of swarm robotics; Rehan, the "director"; James, Tim, Fred and all the people at EPFL. It's been hard, but we made it!

Thanks to the people who worked with me on the research presented in this thesis: Mieke, Diego, Edmond, Michele, Rosario, Francesco.

Thanks to all the IRIDIAns (in random order): Marco M., Sambo, *Borrotti, Alex C., Antal, Michele, Paola, Xico, Alex S., Prasanna, Sara, Matteo, Yara, Stefanie, Jessica, Gabri, Cristina, Alessandra, Tianjun, Manuel, Franco, Bruno, Carlotta, Vito, Saifullah, Jeremie, Sabrina, Eric, Andrea, Gio R., Anthony, Leonardo, Lorenzo, Dj, Roman, Leslie, Gaetan, Tarik, Touraj, Colin, Muriel. Hopefully I didn't forget anybody.

Thanks also to all my friends in Brussels, for making my last years special. Thanks to (in random order): the Lazy Bobos: Julian, Nico, Fanny (best neighbor ever), Anthony; Elisa B., Luisa, Silvio, Stefania, Cate, Fra B., Jorge, Tinna, Annika, Inga, Jacob, Roberto, Andrea, Anna, Alberto, Fra P., Enrico, Marco V., Antoine, Ilaria, my Leuven buddies: Davide, Adamo, Dondi, Damiano, Ann; Claudio, Marta, Camilla, Alex, Anni, Ela, Vivi, Maria, Silvia G. M., Silvia M., Silvia P., Elisa C., Lara, Sachin, Amalia, Paloma, Amandine, Yana, Sanne and you who are reading but I forgot, sorry!

Thanks to all my friends in Italy, I don't make a list because otherwise these acknowledgements will be longer than the thesis itself. Thanks!

Un grazie speciale alla mia famiglia: grazie mamma, Mario e Teo e a tutti gli zii per avermi dato tutto il vostro supporto e affetto anche lontano da casa e per aver creduto in me; senza di voi non sarei mai arrivato fino a qui.

Finally a very special thanks to Elisabetta, I love you.

Contents

Chapter 1

Introduction

The history of mankind has been characterized by the development of tools: since the invention of the stone hand axe, humans have developed aids to carry out more and more complex tasks. Robots, one of the most advanced products of mankind, can be considered as a very particular kind of tools. Similar to the hand axe or the steam engine, each robot has its own purpose: welding, painting, cleaning, etc. However, differently from the majority of tools, robots do not simple aid humans in tackling tasks, but they *replace* them.

Robots now routinely substitute humans in some well-structured tasks, to the point that 60% of tasks in the automotive industry are currently performed by robots, a number which is predicted to raise to 80% by 2050 (Pelez and Kyriakou, 2008). In the coming years, robots will substitute humans also in situations that require the ability to adapt and face unforeseen events and unstructured environments. Autonomous robots will perform more and more complex and delicate tasks in fields as diverse as health care and education, where it is predicted that, by 2050, they will perform 60% and 30% of all tasks respectively (Pelez and Kyriakou, 2008). In the forthcoming years, we will observe the wonder of cars built almost exclusively by robots which will autonomously carry humans to their destinations, where they will cooperate with other robots to perform their everyday jobs.

The ubiquity of robots and their large number will necessarily require every robot to interact and cooperate not only with humans but also with other robots. Robot to robot cooperation provides a pletora of opportunities: multi-robot systems can autonomously execute multiple tasks at the same time and they can combine their abilities to achieve better performance compared to single-robot systems. In this dissertation, we focus on a particular kind of multi-robot systems: robot swarms.

Robot swarms are large groups of robots that operate without relying on any external infrastructure and on any form of centralized control. The collective behavior of a robot swarm is the results of the local interactions between the robots, and between the robots and the environment in which they act. The design of robot swarms is guided by *swarm intelligence* principles, which promote the realization of systems that are *fault tolerant*, *scalable*, and *flexible*. The discipline that studies robot swarms is called *swarm robotics* (Dorigo et al., 2014).

Swarm robotics appears to be a promising approach when different activities must be performed concurrently, when high redundancy and the lack of a single point of failure are desired, and when it is technically unfeasible to setup an infrastructure to control the robots in a centralized way. Some examples of potential applications of swarm robotics are: demining, search and rescue, cleaning, exploration, transportation of large objects, surveillance.

Despite its many potential applications and the potential to promote the realization of systems which are fault tolerant, scalable and flexible, swarm robotics has not been used yet for real-world applications. A deep analysis of the literature on swarm robotics leads us to conjecture that the main current limit of swarm robotics is the lack of an engineering methodology. In other words, in order to tackle real-world applications, we first need a mature *swarm engineering*.

Swarm engineering is the systematic application of scientific and technical knowledge to model and specify requirements, design, realize, verify, validate, operate and maintain a robot swarm.

We maintain that, in order to tackle real-world applications, the priority should be given to developing a systematic approach to *design* robot swarms and a formal way to verify and *analyze* their properties.

In this dissertation, we present some advances in the state of the art in swarm engineering: we propose a novel approach to the design and analysis of robot swarms based on the use of formal methods. In particular, we use model checking, a technique that allows one to formally verify properties of a given model, to design and analyze a robot swarm.

The *design* of robot swarms is challenging. Robot swarms are self-organized systems that have a dual nature: the individual, or microscopic level, and the collective, or macroscopic level. The individual level is the behavior displayed by a single robot. The collective level is the behavior displayed by the swarm and it is the result of the interaction of the individual behaviors.

On the one hand, the dual nature of robot swarms is key in achieving fault tolerance, scalability and flexibility. On the other hand, it is the source of difficult design challenges. The swarm robotics engineer must *think at the collective level, but develop at the individual level*: developers of robot swarms are caught between collective-level missions, and individual-level software. In fact, missions are expressed at the collective level, such as "monitor the perimeter of a building for intruders" or "transport these heavy objects", but the only controllable/programmable components of a robot swarm are the individual behaviors of the robots. Conversely, at the individual level, collective-level missions could be meaningless: for example, for a single robot it is impossible to monitor an entire building at the same time or transport an object if it is too heavy to move. However, the swarm is an "immaterial" concept and thus it cannot be programmed.

Unfortunately, designing and developing the behavior of the individual robots to obtain a desired swarm-level goal is, in general, very difficult, as it is difficult to predict and thus design the non-linear interactions of tens or hundreds individual robots that result in the desired collective behavior.

In system engineering, one common approach is the *divide et impera*[1]: a complex problem is recursively divided into subproblems which are simple to solve; the solutions of each subproblem are then combined together to obtain the solution to the original problem. The divide et impera approach is based on the assumption that solutions can be combined "linearly", that is, modules developed to solve specific subproblems interact with each other in a simple way which is possible to predict precisely. This approach is not feasible for the design of robot swarms: solutions for subproblems interact in a non-linear way, which is generally difficult to predict, often resulting in a system unable to solve the complete problem. For this reason, traditional system engineering approaches are ineffective in swarm robotics (Wooldridge and Jennings, 1998, Banzhaf and Pillay, 2007).

Even general design approaches for multi-robot and multi-agent systems cannot be employed for robot swarms. The design of multi-robot and, more in general, multi-agent systems, has been address in many research papers (Zambonelli et al., 2001, Bordini, 2009, Goldberg and Mataric, 2001). However, the design of robot swarms poses challenges that are not present in multi-agent systems. Indeed, the characteristics of robot swarms, such as high number of individuals, strong decentralization, local communication and action, are usually regarded as characteristics that make a multi-agent system "too complex to

[1]*divide and rule*

manage effectively" (Wooldridge and Jennings, 1998).

Due to the impossibility of using design approaches taken from system engineering or multi-robot systems, other design approaches are necessary.

The most common approach to the design of robot swarms is the bottom-up approach: in a trial-and-error process, the designer develops and improves the behavior of the individual robots until the desired collective behavior is obtained. This approach is not optimal: being a non-structured approach, the quality of the obtained swarm depends completely on the ingenuity and expertise of the designer. Moreover, the trial-and-error process is time consuming and cannot provide guarantees on the obtained results in terms of system correctness and properties. Top-down approaches tackle some of these limits, but, up to now, the developed top-down approaches are very complex and task-specific.

In this dissertation, we present our novel contribution to the top-down design of robot swarms: property-driven design. Property-driven design is based on the concepts of prescriptive modeling and model checking. Using property-driven design it is possible to design robot swarms in a systematic way, realizing systems that are "correct by design".

Developing techniques to *analyze* a robot swarm is also a necessary step in order to employ swarm robotics in real-world applications.

Any system needs to be verified and analyzed before it is used. This is particularly true for systems that interact physically with humans or the environment, as failures and unexpected behaviors could result in damage to people or objects. For this reason, it is necessary to formally analyze any robotics system to verify that it satisfies properties such as safeness and reliability.

Robot swarms are particularly challenging to analyze: due to their self-organized dual nature, robot swarms can be observed and thus analyzed at the macroscopic or microscopic level.

The analyses performed at the two level can be significantly different: the microscopic level allows the research to analyze in details the interactions between robots and between robots and environment. However, this level of detail becomes an obstacle when the size of the swarm is large, as it becomes computationally unfeasible to track the interactions of tens or hundreds of robots. For this reason, very often robot swarms are analyzed at the macroscopic level, considering only the collective behavior of the swarm.

The most common approaches to the analysis of a robot swarms is either using computer simulations or rate equations.

Computer simulations are irreplaceable tools for analyzing the behavior of a robot swarm, as they allow researchers to observe both the individual behaviors of the robots and the collective behavior of the swarm at the same time. However, computer simulations can only validate a small subset of the possible execution scenarios and are impractical to exhaustively verify a collective behavior. In other words, they cannot ensure a complete coverage of the critical aspects of the system nor the absence of anomalies.

Rate equations, instead, are able to analyze the steady-state behavior of a robot swarm, that is, its long-term average behavior. For instance, using rate equations it is possible to ensure that the behavior of the robot swarm converges to a desired objective. However, with rate equations it is difficult to analyze the evolution of the behavior of the robot swarm or any far-from-equilibrium anomalies.

In this dissertation, we explore the use of formal methods, in particular of model checking, to analyze and verify the properties of a robot swarm. Model checking allows us to formally describe a set of desired properties of a system, in a more powerful and precise way compared to other mathematical approaches, and verify whether a given model of a system satisfies them.

In this dissertation, we propose two different approaches: we develop a model of a robot swarm performing collective decision using Bio-PEPA, which allows researchers to define a single high level model, that can then be analyzed using stochastic simulations, rate equations and model checking; we also develop a model of a collective transport behavior using KLAIM, which allows researchers to capture both the hardware and behavior of the robots used.

Contributions and related publications

In this section, we list the contributions presented in this dissertation and list the related scientific publications.

1. We provide a precise definition of swarm robotics and its characteristics. See Chapter 1.

    - Dorigo, M., Birattari, M., and **Brambilla, M.** (2014). Swarm robotics. *Scholarpedia*, 9(1):1463,

2. We perform an in-depth analysis of the state of the art of swarm robotics. Our focus is on swarm robotics as an engineering field. See Chapter 2.

- **Brambilla, M.**, Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

3. We present a novel top-down method for the design of robot swarms based on prescriptive modeling and model checking. See Chapter 3.

    - **Brambilla, M.**, Pinciroli, C., Birattari, M., and Dorigo, M. (2012). Property-driven design for swarm robotics. In *Proceedings of AAMAS*, pages 139–146. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).
    - **Brambilla, M.**, Dorigo, M., and Birattari, M. (2014). Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking. *ACM Transactions on Autonomous and Adaptive Systems*. Submitted for publication.

4. We present two novel ways to use model checking to analyze robot swarms. In particular, we use Bio-PEPA and KLAIM to perform a complete analysis of two robot swarms performing collective decision-making and collective transport. See Chapter 4.

    - Massink, M., **Brambilla, M.**, Latella, D., Dorigo, M., and Birattari, M. (2012). Analysing robot swarm decision-making with Bio-PEPA. In *Swarm Intelligence*, volume 7461 of *Lecture Notes in Computer Science*, pages 25–36. Springer, Berlin, Heidelberg.
    - Massink, M., **Brambilla, M.**, Latella, D., Dorigo, M., and Birattari, M. (2013). On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics. *Swarm Intelligence*, 7(2–3):201–228.
    - Gjondrekaj, E., Loreti, M., Pugliese, R., Tiezzi, F., Pinciroli, C., **Brambilla, M.**, Birattari, M., and Dorigo, M. (2012). Towards a formal verification methodology for collective robotic systems. In *Formal Methods and Software Engineering*, volume 7635 of *Lecture Notes in Computer Science*, pages 54–70. Springer, Berlin, Heidelberg.

5. We analyze a collective transport behavior developed by us for the Swarmanoid project. The behavior can be used to let a group of robots transport an object to a goal area while simultaneously avoid obstacles. This behavior is analyzed in Section 4.2.

- Ferrante, E., **Brambilla, M.**, Birattari, M., and Dorigo, M. (2010). Look out!: Socially-mediated obstacle avoidance in collective transport. In *Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS)*, number 6234 in *Lecture Notes in Computer Science*, pages 572–573. Springer, Berlin, Heidelberg.
- Ferrante, E., **Brambilla, M.**, Birattari, M., and Dorigo, M. (2013). Socially-mediated negotiation for obstacle avoidance in collective transport. In *Proceedings of the International Symposium on Distributed Autonomous Robotics Systems(DARS)*, volume 83 of *Springer Tracts in Advanced Robotics Series*, pages 571–583. Springer, Berlin, Heidelberg.
- Dorigo, M., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., **Brambilla, M.**, Brutschy, A., Burnier, D., Campo, A., Christensen, A., Decugnière, A., Di Caro, G., Ducatelle, F., Ferrante, E., Förster, A., Martinez Gonzales, J., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Rétornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stützle, T., Trianni, V., Tuci, E., Turgut, A. E., and Vaussard, F. (2012). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4):60–71.

Other contributions:

6. We briefly present ARGoS: the first simulator expressly conceived for swarm robotics. All the simulated experiments presented in this dissertation have been conducted using the ARGoS simulator. See Appendix A.4.

   - Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., **Brambilla, M.**, Mathews, N., Ferrante, E., Caro, G. D., Ducatelle, F., Stirling, T., Gutierrez, A., Gambardella, L. M., and Dorigo, M. (2011). ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5027–5034.
   - Pinciroli, C., Trianni, V., OGrady, R., Pini, G., Brutschy, A., **Brambilla, M.**, Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

7. We present a collective behavior performing group size counting. This work is briefly described in Section 2.1.3.

    - **Brambilla, M.**, Pinciroli, C., Birattari, M., and Dorigo, M. (2009). A reliable distributed algorithm for group size estimation with minimal communication requirements. In *Fourteenth International Conference on Advanced Robotics (ICAR)* 2009. Proceedings on CD-ROM, paper ID 137.

8. We used evolutionary robotics to developed a robot swarm able to perform aggregation. The developed behavior was analyzed using rate equations. This work is not presented in this dissertation.

    - Francesca, G., **Brambilla, M.**, Trianni, V., Dorigo, M., and Birattari, M. (2012). Analysing an evolved robotic behaviour using a biological model of collegial decision making. In *Proceedings of the 12th International Conference on Adaptive Behavior (SAB)*, volume 7426 of *Lecture Notes in Computer Science*, pages 381–390. Springer, Berlin, Heidelberg.

Dissertation outline

The rest of the dissertation is organized as follows.

In Chapter 2, we present the background of this dissertation. In particular, in Section 2.1 we present the state of the art in swarm robotics, concerning design methods, analysis methods and collective behaviors; in Section 2.2, we give an introduction to formal methods, focusing on the tools used in this dissertation.

In Chapter 3, we present property-driven design, a design method for swarm robotics based on prescriptive modeling and model checking. We employ property-driven design to develop two robot swarms able to tackle aggregation and foraging.

In Chapter 4, we present two different approaches for the analysis of robot swarms using model checking: in Section 4.1, we use Bio-PEPA to analyze a collective decision-making behavior; in Section 4.2, we use KLAIM to analyze a collective transport behavior.

In Chapter 5, we conclude this dissertation with a summary of the main contributions and some future directions.

Finally, in Appendix A, we describe the simulated and real-robot platforms used in this dissertation.

Chapter 2

Background

In this chapter, we introduce the context in which the research described in this dissertation has been carried out.

In Section 2.1, we present a review of the literature on swarm robotics. In particular, after introducing swarm robotics, we present the literature on design methods, analysis methods and collective behaviors. We conclude with a discussion about the current limits of swarm robotics.

In Section 2.2, we introduce the formal methods that are an essential tool in all the novel design and analysis methods presented in this dissertation. In particular, we introduce model checking together with the model formalism and logic languages adopted to perform model checking.

2.1 Swarm robotics

Swarm robotics (Dorigo et al., 2014) studies how to design large groups of robots that operate without relying on any external infrastructure and on any form of centralized control. In a robot swarm, the collective behavior of the robots results from local interactions between the robots and between the robots and the environment in which they act. The absence of a centralized controller and the great redundancy that characterize a robot swarm promote the realization of systems that are fault tolerant, scalable and flexible.

Different definitions and characterizations of swarm robotics have been proposed and adopted by Sahin (2005), Beni (2005) and Dorigo and Şahin (2004).

Swarm robotics appears to be a promising approach when different activities must be performed concurrently, when high redundancy and the lack of a single point of failure are desired, and when it is technically unfeasible to setup an infrastructure to control the robots in a centralized way. Examples of tasks that could be profitably tackled using swarm robotics are demining, search and rescue, planetary or underwater exploration, and surveillance.

Swarm robotics stems from *swarm intelligence*, the disciplines that studies how simple individuals can self-organize to produce complex and fascinating behaviors both in natural and artificial systems (Bonabeau et al., 1999, Dorigo and Birattari, 2007).

The main source of inspiration for swarm intelligence are social animals. Swarms of ants, colonies of bees, flock of birds and school of fish are some examples of how simple individuals can become successful when they gather in groups. Some examples of collective behaviors of social animals can be found in Figure 2.1.

Desirable properties of robot swarms

The characteristics of swarm robotics encourage the realization of systems that are fault tolerant, scalable and flexible.

Swarm robotics promotes the development of systems that are able to cope well with the failure of one or more of their constituent robots: the loss of individual robots does not imply the failure of the whole swarm. Fault tolerance is enabled by the high redundancy of the swarm: the swarm does not rely on any centralized control entity, leaders, or any individual robot playing a predefined role.

Swarm robotics also enables the development of systems that are able to cope well with changes in their group size: ideally, the introduction or removal of individuals does not cause a drastic change in the performance of the swarm. Scalability is enabled by local sensing and communication: provided that the introduction and removal of robots does not dramatically modify the density of the swarm, each individual robot will keep interacting with approximately the same number of peers, those that are in its sensing and communication range.

Finally, swarm robotics promotes the development of systems that are able to deal with a broad spectrum of environments and operating conditions. Flexibility is enabled by the distributed and self-organized nature of a robot swarm: in a swarm, robots dynamically allocate themselves to different tasks to match the requirements of the specific environment and operating conditions; moreover, robots operate on the basis of local sensing and communication and do not rely on pre-existing infrastructure or on any form of global information.

(a) Ants physically connect one to the other to reach a leaf.

(b) A school of fish forms a ball to protect from a predator.

(c) A swarm of bees protects its hive.

(d) A flock of birds forms a v-formation.

Figure 2.1: Examples of collective behaviors of social animals. Sources: (a) CC BY 2.0 - Kasi Metcalfe - http://www.flickr.com/photos/kasimetcalfe/339113868 (b) CC BY 2.0 - Adam Rifkin - http://www.flickr.com/photos/ifindkarma/8668633683 (c) CC BY 2.0 - Max Westby - http://www.flickr.com/photos/max_westby/1168957201 (d) CC BY 2.0 - Michael Patrick - http://www.flickr.com/photos/michaelpatrick/6470101243

Potential applications of swarm robotics

The properties of robot swarms make them appealing in several potential fields of application.

The use of robots for tackling dangerous tasks is clearly appealing as it eliminates or reduces risks for humans. The dangerous nature of some tasks implies a high risk of losing robots. In these cases, a solution that is fault tolerant is necessary, making dangerous tasks an ideal field of application for robot swarms. Example of dangerous tasks that could be tackled using robot swarms are demining, search and rescue, and toxic cleaning.

Other potential applications for robot swarms are those in which it is difficult or even impossible to estimate in advance the resources needed to accomplish the task. For instance, allocating resources to manage an oil leak can be very hard because it is often difficult to estimate the oil output and to foresee its development. In these cases, it is necessary to adopt a solution that is scalable and flexible. A robot swarm could be an appealing solution: robots can be added or removed in time to provide the appropriate amount of resources and meet the requirements of the specific task. Example of tasks that might require an a priori unknown amount of resources are search and rescue, transportation of large objects, tracking, and cleaning.

Another potential field of application for swarm robotics are tasks that have to be accomplished in large or unstructured environments, in which there is no available infrastructure that can be used to control the robots, such as no available communication network or global localization system. Robot swarms could be employed for such applications because they are able to act autonomously without the need of any infrastructure or any form of external coordination. Examples of tasks in unstructured and large environments are planetary or underwater exploration, surveillance, demining, cleaning, and search and rescue.

Some environments might change rapidly over time. For instance, in a post earthquake situation, buildings might collapse changing the layout of the environment and creating new hazards. In these cases, it is necessary to adopt solutions that are flexible and can react to events. Swarm robotics could be used to develop flexible systems that can adapt to new operating conditions. Example of tasks in environments that change over time are patrolling, disaster recovery, search and rescue, cleaning.

We believe that, to develop robot swarms able to cope with the practical challenge of real-world applications, it is necessary to develop a swarm engineering.

Swarm engineering

Swarm engineering is the systematic application of scientific and technical knowledge to model and specify requirements, design, realize, verify, validate, operate and maintain a swarm intelligence system. Swarm engineering as a term was introduced by Kazadi (2000), who recognized that the focus of swarm intelligence research is moving towards *"the design of predictable, controllable swarms with well-defined global goals and provable minimal conditions"*. He also adds that *"to the swarm engineer, the important points in the design of a swarm are that the swarm will do precisely what it is designed to do, and that it will do so reliably and on time"* (Kazadi, 2000). However, the first work to formally introduce swarm engineering was published only five years later, with the seminal paper by Winfield et al. (2004).

Swarm engineering is still in a very early stage and its development is not homogeneous. On the one hand, some topics, such as design and analysis, have already received attention from the swarm robotics community and several approaches have been proposed. For these topics, our goal is to present and classify the existing works and identify the current limits. On the other hand, other topics, such as requirements analysis, maintenance and performance measurement, have received almost no attention. In Section 2.1.4, we propose a discussion of these topics with the hope to foster new ideas and promote their development.

The outline of this section

In Section 2.1.1, we discuss methods to *design* robot swarms. In Section 2.1.2, we discuss methods to *analyze* robot swarms. In Section 2.1.3, we discuss some of the possible *collective behaviors* a robot swarm can exhibit. By collective behaviors we mean behaviors of the swarm considered as a whole. Such collective behaviors can be used as building blocks in applications, such as foraging or construction (see also Section 2.1.4).

In Section 2.1.4, we conclude with a discussion of the open problems in swarm robotics and swarm engineering.

2.1.1 Design

In this section, we classify the available methods to design robot swarms by dividing them in *manual design methods* and *automatic design methods*.

The difference between these two classes of methods is in the role of the human designer: in the first class, manual design methods, the human designer designs and develops the robot swarms relying on his ingenuity and expertise; in the second class, automatic design methods, the human designers specifies the setup of a computationally intensive process that automatically develops the desired robot swarm.

In the following, we discuss manual design methods and automatic design methods by describing the general principles and the relative advantages and disadvantages.

Manual design methods

In swarm robotics, the most commonly used design method consists in developing, by hand, the individual behaviors of the robots which results in the collective behavior of the swarm.

It is possible to manually develop a robot swarm in two ways: *bottom-up* and *top-down*. In the bottom-up approach, the developer, in a trial-and-error iterative process, implements, analyzes and improves the behavior of the individual robots until the desired collective behavior is obtained. In the top-down approach, instead, the developer performs an analysis of the desired collective behavior and then tries to derive the behavior of the individual robots.

Bottom-up approaches have as their major limit that the quality of the obtained robot swarm completely depends on the ingenuity and expertise of the designer, as the trial-and-error process does not provide any guidance to obtain a quality result. Despite this limit, bottom-up approaches are the most used for the design of robot swarms, mainly because, up to now, there is no general top-down approach.

Bottom-up. Generally, in swarm robotics, the behavior of an individual robot is reactive: a robot takes decisions only on the basis of its sensory inputs and/or its internal memory without planning its actions (Brooks, 1986). This allows developer to use probabilistic finite state machines (PFSMs).

In the context of the PFSMs used in robotics, states represent possible actions of the robot whereas transitions represent conditions on the sensory inputs of the robots; the current state of the PFSM represents the current action performed by the robot. Two states are connected if the two related actions can be performed in a sequence. For example, a robot searching for an object and grabbing it can be represented by the state `search` linked to the state `grab` by the transition `object` `found`. Transitions in PFSMs can be deterministic, as in the previous example, or probabilistic, if a transition has a certain probability to be selected. The transition probability between states can be fixed or can change over time.

The main advantages of using PFSMs are: i) actions and conditions can be easily combined in a structured way; ii) PFMSs can be used recursively, generating behaviors composed of sub-behaviors; iii) PFSMs are easy to read, understand and analyze; iv) PFSMs modularity promotes the reuse of the developed behaviors.

PFSMs have been used to develop several collective behaviors, such as aggregation (Soysal and Şahin, 2005), chain formation (Nouyan et al., 2008) and task-allocation (Liu et al., 2007, Labella et al., 2006). These behaviors will be explained in more details in Section 2.1.3.

Another manual bottom-up approach is *virtual physics*. In virtual physics, each robot is considered as a virtual particle that exerts virtual forces on other robots.

One of the first works using virtual physics-based design was by Khatib (1986), who used the concept of *artificial potential field*. In this and in some following works, the robots are subject to repulsive virtual forces originating from the environment: the goal is associated with an attractive force and the obstacles with repulsive forces. Later, Spears et al. (2004) proposed a complete virtual physics-based design method called physicomimetics framework.

The main advantages of virtual physics-based design methods are: i) a single mathematical rule smoothly translates the entire sensory inputs space into the actuators output space without the need for multiple rules or behaviors; ii) the obtained behaviors can be combined using vectorial operations; iii) some properties (such as fault tolerance, stability, etc.) can be proved using theoretical tools from physics, control theory or graph theory (Gazi and Passino, 2002).

The virtual physics-based method is often used to design collective behaviors that require a robot formation. Examples of such behaviors are pattern formation, collective exploration and coordinated motion (see Section 2.1.1).

Top-down. An alternative approach for the manual design of robot swarms is the top-down approach. In the top-down approach, the developer designs first the collective behavior of the swarm and then derives the behavior of the individual robots. This process is very challenging, as there is no proven methodology to derive individual behaviors from a collective behaviors. Below we present some top-down design approaches that tackle this challenge.

Bachrach et al. (2010) proposed a scripting language called Protoswarm based on the *amorphous computational medium* (Beal, 2004). The amorphous computational medium considers the environment as filled with individuals able both to perform computations and to communicate with their neighbors. Using Protoswarm it is possible to define behaviors for an individual by writing scripts at the collective level. Several collective-level primitives exist in the scripting language, both related to space and time. These primitives are automatically translated into individual behaviors by exploiting the underlying local communication. This language, even though it cannot be considered a standalone design method, can significantly ease the design process thanks to its collective-level primitives. The use of the amorphous computational medium is particularly suited for sensor networks, where the great number of individuals guarantees that the system is able to cover the entire environment with a single connected communication network.

Kazadi et al. (2009) developed a top-down design approach based on Hamiltonian vector fields called the *Hamiltonian method*: starting from a mathematical description of a collective behavior, the method automatically derives microscopic rules that minimize or maximize a selected numerical value (e.g., the virtual potential energy of a particular state of the swarm). The main drawback of the Hamiltonian method is that it deals only with spatially-organizing behaviors, such as pattern formation.

Berman et al. (2009) proposed a top-down approach for the design of a task allocation behavior. The authors describe the system as a Markov chain in which states represent tasks and edges represent the possibility for a robot to move from a task to another. Using a stochastic optimization method, it is possible to derive the probabilities that governs how robots change task in order to minimize the time needed to converge to the desired allocation. This approach is specific for task allocation and it has not been extended to other collective behaviors.

Hamann and Wörn (2008) proposed a method inspired by statistical physics. The authors use Langevin and Fokker-Planck equations to derive the individual behaviors of the robots from the collective behavior of the swarm. A similar approach was used also by Berman et al. (2011a), who used a set of advection-diffusion-reaction partial differential equations to derive the individual behaviors of a swarm performing task allocation. Both methods are based on advanced mathematical techniques and on a detailed model of the interactions of the individual robots, which is usually difficult to realize. Moreover, such methods rely on ordinary or partial differential equations, which provide reliable results only if it is assumed that the swarm size tends to infinite. In real applications of swarm robotics this is very often not the case, since typically robot swarms are composed of no more than a hundred robots and often of just a few tens of robots (Brambilla et al., 2013).

In this dissertation, we propose property-driven design, a novel method for the top-down design of robot swarms. See Chapter 3 for more details.

Automatic design methods

The use of automatic design methods allows the automatic generation of behaviors without the explicit intervention of the developer, avoiding altogether the individual/collective problem.

In the following, we present some works on evolutionary robotics, reinforcement learning and other automatic design methods for swarm robotics.

Evolutionary Robotics. Evolutionary robotics (Nolfi and Floreano, 2000) is an automatic design method that applies evolutionary computation techniques (Goldberg, 1989, Holland, 1975) to single and multi-robot systems. Evolutionary computation is inspired by the Darwinian principle of natural selection and evolution. As such, it is usually presented using a vocabulary borrowed from biology.

Within swarm robotics, evolutionary robotics has been used in many proof-of-concept test-cases in order to test the effectiveness of the method (Baldassarre et al., 2007, Groß and Dorigo, 2008a, Sperati et al., 2008) or as a tool to answer some more fundamental scientific questions (Trianni and Dorigo, 2006, Tuci et al., 2004, Pini and Tuci, 2008, Ampatzis et al., 2008). In this dissertation, we analyze evolutionary robotics from an engineering perspective, that is, we describe its strengths and weaknesses as a design method.

The evolutionary robotics method can be described by the following process. At the beginning, a population of individual behaviors is generated at random. In each iteration, a number of experiments for each individual behavior is executed. In general, the same individual behavior is used by all the robots in the experiment. In each experiment, a fitness function is used to evaluate the collective behavior of the swarm resulting from that individual behavior. At this point, a selection of the highest scoring individual behaviors are modified by genetic operators, such as *cross-over* and *mutation*, and used for the subsequent iterations. See Waibel et al. (2009) for a taxonomy of fitness functions in swarm robotics.

In evolutionary robotics, the evolutionary method is used to find the parameters of an artificial neural network. Although several types of neural networks exist in the literature, they can be roughly categorized in two main classes: feed forward neural networks (Fine, 1999) and recurrent neural networks (Beer and Gallagher, 1992, Elman, 1990). Feed-forward neural networks are used for individual behaviors that require no memory of previous observations and actions. Conversely, recurrent neural networks are used for individual behaviors that require a memory of previously seen input patterns (Ampatzis, 2008).

Evolutionary robotics is a very powerful approach for the design of robot swarms, as it solves the problem of identifying the behavior of the individual robots that results in the desired collective behavior. Nonetheless, evolutionary robotics has many limitations, some of which are: i) evolution is a computationally intensive process, that does not give any guarantees on its convergence to a solution; ii) neural networks are black-box and it is often very difficult to understand their behavior; iii) the high representational power of neural networks results in overfitting, that is, the obtain behaviors do not work properly once instantiated on real robots; iv) from an engineering point of view, the complexity of behaviors currently synthesized through artificial evolution is relatively low and the same results may often be achieved by designing the behavior by hand.

Reinforcement learning. Reinforcement learning has not been extensively studied in swarm robotics as its application presents several issues.

The main issue is the decomposition of global rewards into individual rewards (Wolpert and Tumer, 1999). In fact, usually the performance is evaluated at the collective level, but the behaviors to reward lay at the individual level. This challenging problem is called *spatial credit assignment*. Matarić (1998, 1997) addressed this issue by performing experiments with few robots (2 to 4), using communication or signaling to share the reward.

Additionally to the spatial credit assignment, there are also other issues: i) The *size of the state space* faced in RL problems is huge due to the complexity of the robot-to-robot interactions. ii) The *environment perception is incomplete*. This makes the search of the behavior even more complex (Kaelbling et al., 1998). iii) The *environment*, as seen from the individual robot perspective, is *non-stationary* due to the fact that each robot action is influenced by the actions performed by other robots in the same environment or by changes in the environment itself.

Panait and Luke (2005) conducted an extensive review of the state of the art of automatic design methods for multi-agent and multi-robot systems.

Other automatic design methods. In this section, we outline other works on automatic design that do not belong to evolutionary robotics or reinforcement learning.

In many works, the design of the robot swarm is not completely automatic, as the authors design an individual behavior manually and uses an optimization algorithm to assign some of its parameters. For example, this is the case of ALLIANCE (Parker, 1996), a multi-robot architecture that focuses on the achievement of fault tolerant, robust and adaptive task allocation in a team of robots. Another example can be found in the work of Li et al. (2004), who proposed an algorithm that enables on-line learning of some parameters of the robot behaviors in order to achieve diversity and specialization. The learning algorithm is specifically thought for their application: a stick pulling task.

Other approaches try to overcome the limits of evolutionary robotics by either using alternative optimization algorithms or alternative controller architectures. For example, Pugh and Martinoli (2007) compared the particle swarm optimization against a genetic algorithm for on-line learning parameters for a swarm of robots performing obstacle avoidance. They also defined metrics to measure diversity and specialization, and concluded that particle swarm organization is able to achieve a higher degree of diversity in the swarm. Ferrante et al. (2013b) explored an approach based on grammar evolution to tackle a foraging case study, showing how grammar evolution can promote the evolution of behaviors which are easy to understand and reuse.

Finally, in Francesca et al. (2014) we proposed a new approach to the problem of automatic design: a probabilistic finite state machine is automatically generated by an optimization algorithm combining pre-defined simple behaviors and conditions. A more detailed discussion of how this work can be extended using formal methods is presented in Chapter 5.

2.1.2 Analysis

Analysis is an essential phase in an engineering process. In the analysis phase, the swarm engineer studies whether a general property of the designed collective behavior holds or not. The ultimate goal is to verify that a swarm of robots exhibits the desired collective behavior with the desired properties. Properties of the collective behaviors are usually analyzed by means of models.

Robot swarms can be modeled at two different levels: the individual level, or *microscopic* level, that models the characteristics of the single individuals and the interactions among them; the collective level, or *macroscopic* level, that models the characteristics of the swarm as a whole. The development of models for analyzing robot swarms at both levels of abstraction is still a subject of study and research. In fact, modeling both the microscopic and the macroscopic level and their interaction is very difficult due to the nature of self-organized systems (Abbott, 2006). As a consequence, the vast majority of modeling techniques that are used nowadays focus on one level at a time.

The main analysis technique used in this dissertation is model checking applied to Markov chains. For the sake of completeness, in this section, we present some analysis approaches that used Markov chains and model checking, but we postpone a full discussion on this techniques to Section 2.2.1.

We classify the literature on analysis according to whether the main concern is to capture the microscopic or the macroscopic aspects. We conclude with an overview of how the analysis with real robots is conducted.

Microscopic models

Microscopic models focus on the behavior and state of each robot individually, describing both robot-to-robot and robot-to-environment interactions. This level of detail allows us to analyze a robot swarm thoroughly.

The level of detail considered in microscopic models can vary greatly: simplest models consider the robots as point-masses; intermediate-complexity models consider 2D worlds with kinematic physics; more complex models consider 3D worlds with dynamic physics where the details of each sensor and actuator are modeled. For a complete analysis of the different levels of abstraction see Friedmann (2010).

As said, microscopic models give us a very detailed view of a swarm. However, this high level of detail is also the source of the main issue with microscopic models: limited scalability. In order to describe the behavior of a swarm it is necessary to replicate one description of a single individual by the number of individuals in the swarm. This results in a model with a large number of components, which is computationally heavy to treat. For this reason, microscopic models are usually analyzed via computer simulation.

Simulations are among the most used tools to analyze robot swarms. The possibility to simulate in details robot swarms comes at a cost: performing a complete analysis of the robot swarm becomes virtually impossible, as it is impossible to analyze all possible executions of a system. For this reason, it might happen that some rarely occurring problems are not found using simulations.

Simulators employed in swarm robotics have many characteristics in common with simulators employed to study other mobile robotics systems. However, general-purposes robotics simulators usually do not cope well with large number of robots, as scalability is not their main concern. Vaughan (2008) proposed a benchmark to study scalability in multi-robot simulators and applied it to the Stage simulator. In Pinciroli et al. (2012), we developed a simulator for swarm robotics by focusing explicitly on the scalability issue. The developed simulator was able to simulate 10^5 robots in real time. For a survey of various simulation platforms in robotics see Kramer and Scheutz (2007).

Only few authors have focused on microscopic models that do not involve simulations. A recent example is the work of Dixon et al. (2011), who first modeled the behavior of the individual robots as a Markov chain and then modeled the behavior of the swarm as the *and*-composition of individual-level models. They finally applied complete model checking to this model by using linear temporal logic to define properties of individual robots and of the swarm. This approach is not scalable, as the number of states of the model increases exponentially with the number of robots preventing the possibility to analyze swarms larger than a few individuals. Furthermore, linear temporal logic does not allow for the probabilistic quantification of properties, which is an important feature to analyze stochastic systems such as robot swarms. In Section 2.2.2, we present the stochastic temporal logic used in this dissertation.

In Gjondrekaj et al. (2012), we analyzed a collective transport collective behavior at the microscopic level. In this work, we modeled the hardware components of the robots to understand in details their interactions with each other and with the other robots. This approach suffers from the same scalability limits as the one presented by Dixon et al. (2011): the model is too detailed to analyze a large robot swarm. However, in this case we focus on collective transport, a task involving a limited number of robots; moreover we employ statistical model checking, which allows us to partially overcome the scalability issues. More details can be found in Section 4.2.

Macroscopic models

Macroscopic models consider robot swarms as a whole. The individual robots of the swarm are not taken into account, in favor of a description of the swarm at a higher level. Macroscopic models are the most commonly used models for the analysis of robot swarms.

In this section, we provide a broad overview of the main contributions in this area. We classify works on macroscopic modeling into four categories. In the first category, we consider works resorting to rate or differential equations. In the second category, we consider works employing Langevin and Fokker-Plank equations. In the third category, we present works where classical control and stability theory are used to prove properties of a robot swarm. In the last category, we consider other approaches.

Rate and differential equations. Rate equations are ordinary differential equations (ODE) which are used to describe the time evolution of the behavior of the robot swarm. The analysis performed using rate equations is also known as fluid flow analysis (Zarzhitsky et al., 2005) or steady-state analysis (Lerman et al., 2001).

The macroscopic model used is usually derived from a model of the behavior of individual robots, such as the probabilistic finite state machine used for the design of such behavior. More in details:

i) First, a set of variables is defined. Usually, one variable is defined for each state of the individual-level probabilistic finite state machine. These variables are used to track the proportion of robots that are in the corresponding states.

ii) Second, for each variable, a rate equation is defined. This equation describes the instantaneous incoming and outgoing flow of robots to the state to which the variable is associate. This flow is given typically as a function of the number of robots in the preceding and following states. The rate equation contains a set of parameters, at least one for each input and output transition of the corresponding state. Numerically, these parameters can be derived either from the description of the system or empirically.

The rate equations method has been used to model many robot swarms. In their seminal work, Martinoli et al. (1999) used rate equations to model a clustering behavior. Lerman et al. (2001) and Martinoli et al. (2004) used rate equations to model a stick pulling experiment, in which two robots need to cooperate in order to pull sticks out of their holes. Lerman and Galstyan (2002) modeled foraging under the effect of interference. In this case, the authors were able to model the individual performance in foraging as a decreasing function of group size. Trianni et al. (2002) used rate equations to model a chain formation behavior and an aggregation behavior, implemented using probabilistic finite state machines. Campo and Dorigo (2007) modeled the collective behavior of robots performing foraging in an environment containing more than one food source. Winfield et al. (2008) used rate equations to model a swarm of robots whose goal is to stay together while avoiding collisions. Liu and Winfield (2010) used rate equations to model foraging involving the collection of energy units. Finally, Pinciroli et al. (2013) used rate equations to model an aggregation collective behavior. In this work a flying robot can actively control the number of robots aggregating beneath it.

Rate equations are a very powerful technique to analyze a robot swarm. However, they suffer from some limitations:

- in the general case, robot positions in space are not explicitly modeled and actions are assumed to have all the same duration. This hinders the analysis of systems that are strongly characterized by spatial or temporal aspects. Task-specific solutions to this problem have been proposed (Galstyan et al., 2005);
- rate equation can only describe the steady-state behavior of the swarm, not its far-from-convergence behavior;
- the reliability of the obtained results depends on the size of the swarm considered, when swarms composed of less than hundreds or thousands individuals are considered, the results are in general poor.

Langevin and Fokker-Plank equations A recent advancement in macroscopic modeling based on differential equations is due to Hamann and Wörn (2008), who introduced the Langevin and Fokker-Plank equations, both borrowed from the statistical physics literature, to the swarm robotics community.

The Langevin equation is a stochastic differential equation that describe the motion of a particle in a fluid. The Langevin equation can be used to define a "mesoscopic" model (intermediate level between micro and macro). In fact, the motion of the particle is modeled using two components: a deterministic component, that represents the microscopic laws of motion of that particle, and a stochastic component, that represents the interaction of the particle with the environment (in this case the ensemble of particles composing the fluid). In the swarm robotics context, the deterministic component of the Langevin equation models the deterministic motion of the robot controlled by its individual behavior, whereas the stochastic part models the interaction of the robot with the other robots, considered as a flow, and with the environment.

From the Langevin equation it is possible to derive the Fokker-Plank equation. The Fokker-Plank equation can be used to describe the dynamics of the entire swarm. It models the time-evolution of the probability density function that describes the state, such as the position or the velocity, of the all robots in the environment. The derivation of the Fokker-Plank equation starting from the Langevin equation is possible using tools of statistical mechanics plus some problem-dependent intuition.

Hamann and Wörn (2008) applied this modeling method to analyze coordinated motion, aggregation and foraging. Recently, the authors modeled aggregation in presence of a temperature gradient in the environment, and provided a comparison with another model called Stock & Flow (Schmickl et al., 2009).

A similar approach was adopted also by Berman et al. (2009), who used a set of advection-diffusion-reaction partial differential equations to derive the individual behaviors of a swarm performing task allocation. In Berman et al. (2011c), this approach is applied to an area coverage behavior. Dantu et al. (2012) compared the results obtained in this work with those obtained from a simulation of the same behavior. The goal of the authors was to understand the effects of noise and errors on the collective behavior.

Another interesting study on the use of the Fokker-Plank equation was done by Prorok et al. (2011). In their work, the authors compared four different models of an area coverage behavior. They did it by measuring the area covered by the robots. Each model is characterized by being microscopic or macroscopic, and spatial or non-spatial. The predictions obtained from the models are compared with the results of both simulated and real-robot experiments. The authors showed that predictions of the spatial and non-spatial models differ for short time spans, for which the results of spatial models are more accurate, but are very similar for long time spans.

The Fokker-Plank equation approach has the advantage that it can be used, in principle, to model any swarm robotics collective behavior. There are, however, some limits: i) the Fokker-Plank equation is difficult to be solved analytically and sometimes requires computationally demanding numerical algorithms; ii) communication aspects, at present, are very difficult to model; iii) similar to rate equations, to obtain accurate results, only large swarms can be considered; iv) the development of the Langevin and Fokker-Plank equations is very difficult, as it is usually difficult to model in a precise mathematical way the individual-to-individual and individual-to-swarm interactions.

Classical control and stability theory. Some works use classical control and stability theory to prove properties of the swarm. Liu et al. (2003) and Gazi and Passino (2005) modeled a swarm of agents in a one-dimensional space using discrete-time discrete-event dynamical systems. Liu and Passino (2004) and Gazi and Passino (2004b) used Lyapunov stability theory to prove that the behavior studied was able to let a swarm achieve coherent social foraging in presence of noise. Similarly, Gazi and Passino (2003, 2004a) proved that, in specific conditions, a swarm of agents aggregates in one point of the environment. Schwager et al. (2011) modeled a swarm of communicating robots as a linear, discrete-time dynamical system. The authors then used their model and Lyapunov stability theory to study how different communication topologies affect the stability of the system. Finally, Hsieh et al. (2008) used delay differential equations to model task-allocation (agents allocating and re-allocating to different physical sites), proving the stability of the reached configuration. In the same work, the authors also proposed a method to compute the optimal transition matrix in order to obtain a swarm that reaches the desired configuration.

All these modeling methods have the advantage to be based on strong mathematical formulations. However, the main problem with these methods is that they usually ignore characterizing aspects of swarm robotics, such as asynchronicity, stochasticity and the absence of global information.

Other modeling approaches. In the third and final category we consider works in modeling that resort to other mathematical frameworks.

Soysal and Şahin (2007) modeled aggregation using Markov chains and validated the prediction using simulation.

The work of Turgut et al. (2008b) represents one of the first modeling attempts to bridge studies of flocking within physics with studies of flocking within robotics. In their study, the authors modeled alignment in flocking. The model shows that there is a phase transition from ordered flocking to non-ordered flocking corresponding to a critical value of noise in the sensor used to perceive the heading of other robots. The results were validated using simulation.

Correll (2008) used a population dynamics model to find the parameters used in two task-allocation behaviors. Using the model and an optimization algorithm, the authors could estimate the parameters that lead to the optimal distribution of robots.

Mathews et al. (2010) modeled a problem in which a flying robot selects one mobile robot within a group to establish a communication channel with it. To model the swarm, the authors used the theory of branching processes (Kendall, 1966).

Hamann (2012) developed two simple models for robot swarms. In the first, the performance of a generic robot swarm is explained using the interaction between cooperation and interference. In the second, the consensus achievement behavior is studied using a simple probabilistic model based on the urn problem.

Konur et al. (2012) analyzed a robot swarm by using model checking. In particular, they developed a macroscopic Markov chain model of a robot swarm performing foraging and subsequently analyzed its properties using model checking. For more details about model checking, which is central in this dissertation, see Section 2.2.3.

In Massink et al. (2013), we introduced the use of Bio-PEPA to analyze consensus achievement in a robot swarm. Bio-PEPA is a high level modeling language. From a description of a system formulated in Bio-PEPA, one can automatically derive different models apt to perform stochastic simulation, fluid flow (ODE) analysis, and model checking. These models are guaranteed to be consistent. This work is presented in more details in Section 4.1.

Real-robot analysis

The ultimate goal of swarm robotics is to produce swarms of real robots. Despite the amount of sophisticate analysis methods, some properties of these swarms cannot be reliably assesses neither with mathematical models nor with simulations. In fact, it is practically unfeasible to simulate all the aspects of reality (Frigg and Hartmann, 2012, Brooks, 1990). Experiments with real robots help to test the robustness of collective behaviors against noisy sensors and actuators. The transfer from simulation to real robots is particularly relevant in complex systems such as swarm robotics: small differences between simulations and reality could lead to widely diverging behaviors.

It must be noted that in all real-robot experiments presented in the analyzed literature, the experiments are performed in controlled environments. By controlled environments we mean artificial arenas in which most conditions—e.g., light intensity, radio interference and floor smoothness—can be controlled by the experimenter. This is often very far from the scenarios in which robot swarms are supposed to operate. For this reason, real-robot experiments should not be considered as a way to validate collective behaviors for their use in real-world applications, but rather as a way to test them against realistic noise patterns in sensors and actuators.

In Section 2.1.3, we present more than sixty publications dealing with collective behaviors in the swarm robotics field. Slightly more than half of these publications presented results obtained only through simulations or models. We believe that the reason behind this choice is that, in general, it is easier, faster and safer to perform experiments using models or simulations than using robots.

In the papers that included experiments with real robots, the scope of the use of the robots can be divided in two categories: proof-of-concept experiments and extensive experiments. The first category includes slightly more than half of the analyzed works that involve real-robot experiments. In these works, few runs (typically one) of an experiment are performed with real robots. The aim of real-robot experiments within these works is to show that the proposed collective behavior is realizable. Examples of this kind of experiment can be found in the works by Payton et al. (2001) and Spears et al. (2004). In the other category, instead, several runs are executed and data is gathered to be analyzed for comparison with simulated runs or to show properties of the swarm. Examples of this kind of experiments can be found in the works by Çelikkanat and Şahin (2010) and O'Grady et al. (2010).

2.1.3 Collective behaviors

In this section, we present a review of the main collective behaviors studied in the swarm robotics literature. These collective behaviors are basic swarm behaviors that could be combined to tackle complex real-world applications as, for example, foraging or construction. We classify these collective behaviors into four main categories: spatially-organizing behaviors, navigation behaviors, collective decision-making and other collective behaviors.

In the first category, spatially-organizing behaviors, we consider behaviors that focus on how to organize and distribute robots and objects in space. In the second category, navigation behaviors, we consider behaviors that focus on how to organize and coordinate the movements of a swarm of robots. In the third category, collective decision-making, we consider behaviors that focus on letting a group of robots agree on a common decision or allocate among different parallel tasks. In the last category, other collective behaviors, we consider behaviors that do not fall into any of the categories mentioned above.

For each category, we give a brief description of the collective behavior, its source of inspiration, the most common used approaches and the most significant available results.

Spatially-organizing behaviors

In this section, we describe collective behaviors that focus on how to organize and distribute robots and objects in space. In the following, we present works on aggregation, pattern formation, chain formation, self-assembly and morphogenesis, and object clustering.

Aggregation. The goal of *aggregation* is to group all the robots of a swarm in a region of the environment. Despite being an apparently simple collective behavior, aggregation is a very useful building block, as it allows a swarm of robots to get sufficiently close one another so that they can interact.

Aggregation is a very common behavior in nature. For example, aggregation can be observed in bacteria, cockroaches, bees, fish and penguins (Camazine et al., 2001). Other examples of natural systems performing aggregation have been described by Grünbaum and Okubo (1994), Breder Jr (1954), Jeanson et al. (2005), Amé et al. (2006).

The most common design approach used to obtain aggregation is based on probabilistic finite state machines: the robots explore an environment and, when they find other robots, they decide stochastically whether to join or leave the aggregate. In this approach, a stochastic component is often used in order to promote the formation of a single aggregate. Other studies focus on the use of evolutionary robotics to obtain aggregation.

Garnier et al. (2005) developed a robot swarm to replicate the behavior observed in cockroaches by Jeanson et al. (2005). The robots are able to collectively aggregate in a circular arena using a PFSM approach.

Another example of an aggregation behavior based on a PFSM was developed by Soysal and Şahin (2005, 2007). In their work, a robot can be in one of three states: the repel state, in which the robot tends to get away from other robots; the approach state, in which the robot tends to get closer to other robots; and the wait state, in which the robot stands still. Soysal and Şahin were able to achieve both moving and static aggregation behaviors by changing the parameters of the collective behavior.

An example of aggregation obtained with artificial evolution was developed by Trianni et al. (2003). The authors obtained two sets of parameters for a neural network achieving both moving and static aggregates.

Soysal et al. (2007) presented some rules of thumb for obtaining aggregation behaviors through artificial evolution. Moreover, they proposed a comparison between the probabilistic finite state machine approach by Soysal and Şahin (2005) and the artificial evolution approach by Bahçeci and Şahin (2005).

Pattern formation. *Pattern formation* aims at deploying robots in space in a regular and repetitive manner. Robots usually need to keep specific a distance between each other in order to create a desired pattern.

Pattern formation can be found both in biology and in physics. Some biological examples are the spatial disposition of bacterial colonies and the chromatic patterns on some animal's fur (Meinhardt, 1982). Some physics examples are molecules distribution and crystal formation (Langer, 1980), and Bénard cells (Getling, 1998).

The most common way to develop pattern formation behaviors in robot swarms is to use virtual physics-based design. Virtual physics-based design uses virtual forces to coordinate the movements of robots.

Bahçeci et al. (2003) presented a review of works on pattern formation in which they analyzed centralized and decentralized behaviors. Another review on the topic has been published in 2009 by Varghese and McKee.

Spears et al. (2004) developed a collective behavior for pattern formation that is one of the first applications of virtual physics-based design. In their work, they use the virtual forces to form an hexagonal lattice. In the same work, Spears et al. showed that, by creating two groups of robots with different attraction/repulsion thresholds, it is also possible to obtain a square lattice. More details can be found in a subsequent work (Spears and Spears, 2012).

Shucker and Bennett (2007) presented a behavior in which robots interact via virtual springs. These virtual springs are used by a robot to compute attraction/repulsion virtual forces. Differently from Spears et al.'s work, in this work, the robots can interact in different ways (full connectivity, first neighbors, N-nearest, . . .). Each type of interaction has different characteristics and gives rise to different patterns. Additional theoretical work is presented in a subsequent paper (Shucker et al., 2008).

Flocchini et al. (2008) focused on a theoretical analysis of pattern formation. The authors were able to formally prove that with a robot swarm some patterns are achievable only with some kind of global knowledge such as a common orientation given by a compass.

Chain formation. In the *chain formation* behavior, robots have to position themselves in order to connect two points. The chain that they form can then be used as a guide for navigation.

The chain formation behavior takes its inspiration from foraging ants. Deneubourg et al. (1990) studied and modeled the behavior of Argentine ants, which form chains of individuals connecting their nest with foraging areas.

Chains of robots can be obtained in multiple ways: the most used design approaches are probabilistic finite state machines, virtual physics-based design and artificial evolution.

Nouyan et al. (2008, 2009) developed a behavior, based on probabilistic finite state machines, in which the robots have two different exchangeable roles: explorer and chain member. In the explorer role, the robots are searching for chain members or for the goal area. When they find either a chain member or the goal, they switch to the chain member role and stop. Chain members can become explorer again according to a probability that increases over time if no other robot is perceived. Different configurations and approaches are analyzed and presented.

Maxim et al. (2009) used virtual physics-based design to form chains of robots. Virtual forces are used to keep a specific distance between robots and between a robot and the walls of the environment. The developed behavior creates chains that are strongly based on the shape of the environment, which is assumed to be composed of narrow corridors.

Sperati et al. (2011) used artificial evolution to obtain a chain formation behavior. In their work, the robots, by using communication through colored LEDs, are able to follow each other forming a double chain between two designated areas. Differently from other chain formation behaviors, in this work the obtained chain is composed of moving robots.

Self-assembly and morphogenesis. In robotics, *self-assembly* is the process by which robots physically connect to each other. Self-assembly can be used for different purposes. For example, to increase stability when navigating on rough terrains or to increase the pulling power of the robots. *Morphogenesis* is the process that leads a swarm of robots to self-assemble following a particular pattern, and can be used by the swarm to self-assemble into a structure that is particularly appropriate for a given task. For example, a line formation can allow to pass on a narrow bridge, while a blob-like formation will make moving on rough terrain more stable.

Self-assembly can be observed in several species of ants. Ants are able to physically connect in order to perform different tasks. Some examples of structures created by ants are bridges, rafts, walls and bivouacs (Anderson et al., 2002). Self-assembly and morphogenesis are studied also by developmental biology: scientists study how cells develop and self-organize to form tissues and organs (Turing, 1953).

From the swarm robotics perspective, there are two main challenges: how to self-assemble into a desired target structure (i.e., morphogenesis), and how to control the obtained structure to tackle specific tasks. Works focusing on the first issue are usually based on probabilistic finite state machines and rely on communication for coordination. Works focusing on the second issue, make use either of artificial evolution or of probabilistic finite state machines.

A review of the literature on self-assembly and morphogenesis has been presented by Groß and Dorigo (2008b). Here, we discuss only some examples of recent works.

O'Grady et al. (2009) presented a morphogenesis behavior for self-assembling robots that are able to signal docking points on their body to other robots using LEDs. Different structures, such as lines, stars and circles, can be obtained by having the robots signal docking points in different positions. A scripting language for the morphogenesis process has been presented by Christensen et al. (2008). Both works were realized in the context of the Swarm-Bots project (Dorigo et al., 2006).

Results on the control aspect of self-assembly depend strongly on the goal of the specific swarm. O'Grady et al. (2010) demonstrated that physically connected robots can navigate through difficult terrains better than robots that are not connected. In O'Grady et al.'s work, robots randomly explore an environment with slopes. Each robot is able to measure the steepness of these slopes and when a slope is steeper than a certain threshold, it can initiate a self-assembling procedure. Once connected into a structure, the robots can navigate in hazardous terrains thanks to the high mechanical stability given by the new morphology. Mondada et al. (2005) showed that physically connected robots are able to cross a ditch that is too large for a single robot to overcome. Finally Groß and Dorigo (2009) showed that physically connected robots are able to obtain better results, in terms of speed and distance, in the transportation of heavy objects when compared to non-connected robots.

The Symbrion and Replicator projects tackled both the morphogenesis and the control aspects of self-assembly (Levi and Kernbach, 2010). In these projects, swarms of self-assembling robots capable of creating 3D structures are studied. Such robots are able, when connected, to share energy and computational resources with their neighbors.

Another aspect of self-assembly is how to make the swarm decide who should assemble with whom. In the work of Ampatzis et al. (2009), two robots have to assemble to each other without prior knowledge of who will grip and who will be gripped. The authors proposed a solution based on artificial evolution and recurrent neural networks which can make time-dependent decisions.

Mathews et al. (2012) used a heterogeneous approach: a flying robot is used to recognize the task to tackle and guide ground-based robots. The flying robot communicates to the ground based robots which robots should self-assemble and what kind of structure to create to tackle the task.

Object clustering and assembling. Here, we present works in which robots move objects spread in an environment. The robots can follow two kinds of behaviors: clustering and assembling. The goal of *object clustering and assembling* is to group objects close one to the other. The difference between clusters and assembles is that clusters are composed of non-connected objects, whereas assembles are composed of physically linked objects. The object clustering and assembling behaviors are fundamental components of any construction process.

The object clustering and assembling behaviors are displayed by many social insects. For example, ants exhibit brood clustering (Franks and Sendova-Franks, 1992) and termites are able to deposit mud to build complex nests (Grassé, 1959). To do this, insects usually exploit natural occurring gradients, such as temperature gradients, or pheromones gradients.

In swarm robotics, object clustering and assembling are usually approached using probabilistic finite state machines. The robots explore the environment at random and react in different ways to the discovery of available objects or of part of the cluster/assemble to create.

In almost all analyzed works, robots group objects sequentially or quasi sequentially. In fact, parallelism could potentially create collisions and interference and thus is usually avoided. To avoid such problems, a robot usually prevents other robots from depositing objects at the same time, either by using communication or by physically blocking access to the cluster site.

Object assembles are usually obtained using blocks with some kind of self-alignment mechanism based, for example, on magnets.

One of the pioneering work in object clustering is the one by Beckers et al. (1994). In this work, the robots follow a very simple behavior: they explore the environment at random and, when they find an object, they pick it up. A robot with an object moves at random in the environment and deposits the object with a probability proportional to the number of other objects observed. Following these simple rules, the robots are able to create clusters of objects.

Melhuish et al. (1999b) presented a work in which robots create clusters of object roughly in the shape of a wall. In this work, disks are scattered around the environment and a specific area is marked for disk clustering. Such area is located half way between a light and a line on the ground. The robots measure the distance between the line on the ground and the light to recognize the clustering area. The robots follow simple rules to cluster the object roughly in the shape of a wall. In a following work, Stewart and Russell (2006) developed a similar mechanism in which the position of the cluster is marked by a moving robot instead of a fixed light.

Wawerla et al. (2002) developed a behavior to create simple 2D walls made of blocks of alternating color. A robot performs random walk in search for a block. After collecting a block, the robot searches either for the seed block or for the partially constructed wall. When it finds the wall or the seed block, the robot checks if no other robot is already placing a block by using local communication, and then places the blocks.

Werfel (2006) developed a method for creating arbitrary 2D structures with blocks placed over a virtual grid. In this work, all robots have knowledge of a matrix which encodes the final structure to create. Such plan is used by the robots also as a frame of reference. The idea is the following: when a robot finds part of the structure, it follows it counting the placed blocks. Block-counting allows the robot to locate itself in the frame of reference of the structure to create.

Once the robot knows its position with respect to the placed blocks, it can decide where to place the next block using its knowledge of the final structure. This approach was also extended to create 3D structures (Werfel and Nagpal, 2008) and tested using real robots (Werfel et al., 2011).

Navigation behaviors

In this section, we describe collective behaviors that cope with the problem of coordinating the movements of a swarm of robots. We review works on collective exploration, coordinated motion and collective transport.

Collective exploration. Here, we analyze two kinds of collective behaviors that, together, can be used to achieve *collective exploration* of an environment: *area coverage* and *swarm-guided navigation*. The goal of *area coverage* is to deploy robots in an environment in order to create a regular or irregular grid of communicating robots. The obtained grid can be employed, for example, to monitor the environment for hazardous leaks or to guide other robots. We call the behavior necessary to guide the navigation of other robots *swarm-guided navigation*. Since the two behaviors are strongly linked, many works focus on both at the same time.

Area coverage and navigation are common behaviors of social animals. For example, ants use pheromones trails to find the shortest route between two points and bees directly communicate destinations in the environment by means of dances (Camazine et al., 2001). Area coverage has been intensively studied also by the *wireless sensor networks* community. A survey of area coverage behaviors for wireless sensor networks was conducted by Wang et al. (2009).

In swarm robotics, the most common way to tackle area coverage is to use virtual physics-based design to obtain a grid covering the environment. Works on swarm-guided navigation instead focus on communication, thus usually employ probabilistic finite state machines and take inspiration either from network routing protocols or natural systems.

Payton et al. (2001) used robots as "virtual pheromones". Some robots, which are already deployed, are able to create a gradient between the source and the target by exchanging messages. This gradient can then be exploited for navigation by other robots or by a human.

Howard et al. (2002) developed a behavior using virtual physics-based design. Each robot is repelled by other robots and by obstacles. This approach allows the robots to maximize the area covered and form a connected communication network.

O'Hara and Balch (2007) presented a behavior that exploits pre-deployed sensors to perform foraging in an environment that can change over time. Through a distributed Bellman-Ford algorithm, a navigation route towards a specific goal is found and then followed by the mobile robots.

Nouyan et al. (2009) used chain formation to link an object in the environment with the robot nest. The robots in the chain display a pattern of repeating colors to indicate in which direction is the nest and in which direction is the object. This information is used by the other robots to navigate the environment.

Di Caro et al. (2009) presented a work in which robots are able to navigate from a source to a target location. The proposed behavior is based on communication with other passive robots already available in the environment. These passive robots are assumed busy with other collective behaviors but are able to guide the navigating robots.

Stirling and Floreano (2010) used a swarm of flying robots to achieve area coverage. In their work, the robots are deployed sequentially and each robot determines its position according to the position of the previously deployed robots. Only one or few robots, called explorers, are flying at the same time, whereas the great majority is attached to the ceiling and act as communication relays. One particular aspect of Stirling and Floreano's approach is the ability to explore an environment with a limited number of robots, as the robots can leave an area once it has been visited. This work was developed for the Swarmanoid project (Dorigo et al., 2012).

Ducatelle et al. (2011a) proposed a collective behavior based on network routing, capable of guiding a robot from a source area to a target. Similarly to what happens in packet routing, the robots keep a table of the distance of other robots with respect to the target. A robot can then use the entries in the table and reach the target. Ducatelle et al. studied two different experimental setups. In the first, only a single robot is moving while the others act as beacons to guide it. In the second, all the robots are moving between two points, creating a dynamic chain.

Ducatelle et al. (2011b) studied collective exploration using a heterogeneous robotic swarms. They tackled an indoor navigation task, in which a swarm of wheeled robots move back and forth between a source and a target location. The path of the wheeled robots are guided by a swarm of flying robots that can attach to the ceiling and overview the progress of the wheeled robots. Their solution is based on mutual adaptation: wheeled robots execute instructions given by flying robots, and flying robots observe the behavior of wheeled robots to adapt their position and the instructions they give. Ducatelle et al. developed the swarm using a probabilistic finite state machine together with techniques from network routing protocols.

Coordinated motion. In *coordinated motion*, also known as *flocking*, robots move in formation similarly to schools of fish or flocks of birds. For a group of autonomous robots, coordinated motion can be useful as a way to navigate in an environment with limited or no collisions between robots and as a way to improve the sensing abilities of the swarm (Kaminka et al., 2008).

Coordinated motion behaviors are frequent in almost all social animals. In particular, flocking in group of birds or schooling in group of fish are impressive examples of self-organized coordinated motion (Okubo, 1986). Through coordinated motion, animals gain several advantages, such as a higher survival rate, more precise navigation and reduced energy consumption (Parrish et al., 2002).

In swarm robotics, coordinated motion behaviors are usually based on virtual physics-based design. Robots are supposed to keep a constant distance from one another and an uniform alignment while moving (Reynolds, 1987). Coordinated motion behaviors have also been obtained via artificial evolution.

The first work on coordinated motion was published by Reynolds (1987) in the domain of computer graphics. Reynolds developed a flock of virtual birds in which the individuals are able to sense the velocity and the range of the neighbors. The individuals follow three simple rules: *collision avoidance*, *velocity matching* and *flock centering*. Collision avoidance keeps the individuals from colliding one with the other. Velocity matching ensures that each individual matches the speed of its neighbors and flocking centering forces each individual to stay close to its neighbors.

Balch and Hybinette (2000) proposed a coordinated motion behavior based on social potentials. Each robot knows the position and orientation of the robots in its sensing range and thus it is able to compute the target position to reach. The authors created a coordinated motion behavior that is able to avoid obstacles and form different patterns, such as lines, diamonds and squares.

Baldassarre et al. (2003) used artificial evolution to tune the parameters of a neural network in order to perform coordinated motion. The authors were able to obtain three coordinated motion behaviors. These behaviors differ by how each robot moves with respect to the others. In the first behavior, the robots keep a constant speed. In the second one, only one robot moves, while the rest of the swarm tries to remain close to it. In the last behavior, the robots rotate around the center of the swarm.

Turgut et al. (2008a) developed a virtual heading sensor which allows each robot to sense the heading direction of the other robots. With this information and knowing the distance of the neighbors by means of an infrared sensor, the swarm was able to obtain coherent coordinated motion and obstacle avoidance in absence of a common goal direction. The developed behavior is one of the first true implementation of Reynolds (1987)'s flocking behavior with real robots. The authors evaluated the performance of their behavior by using different metrics and validated it with the use of several robots.

Çelikkanat and Şahin (2010), extending the work of Turgut et al. (2008a), showed that it is possible to insert some "informed" robots in the swarm in order to direct the movement of other "non-informed" robots. The informed robots are the only ones in the group with knowledge of the goal direction. Increasing the number of informed robots or decreasing the individual tendency to follow other robots increase the accuracy of motion of the group with respect to the desired goal direction. These works have been extended by Ferrante et al. (2010b) who developed alternative communication strategies in which some robots explicitly communicate their headings.

Stranieri et al. (2011) first introduced the idea of coordinated motion without the need for all robots to perceive the orientation of their neighbors. Their work has been extended by Ferrante et al. (2012), thus we present in detail only this most recent work.

Ferrante et al. (2012) proposed a coordinated motion behavior that, differently from other works, does not require an explicit alignment rule: the robots in the swarm use only attraction and repulsion rules. The key difference is in the novel way in which the robots translate the vector computed using the attraction and repulsion rules into wheel actuation. In fact, in all previous works, the robots changed their angular speed according to the direction of this vector while keeping their forward velocity fixed. In this work instead, the robots change also their forward speed, according to the magnitude of the computed vector. Additionally, the authors showed that the swarm is able to navigate both with and without the presence of informed robots, that is, robots that know the desired direction to follow.

Collective transport. *Collective transport*, also known as group prey retrieval, is a collective behavior in which a group of robots has to cooperate in order to transport an object. In general, the object is heavy and cannot be moved by a single robot, making cooperation necessary. The robots need to agree on a common direction in order to effectively move the object towards a target.

Ants often carry prey cooperatively. Kube and Bonabeau (2000) analyzed how cooperative transport is achieved in ant colonies. When ants find their target, they physically attach to it and then start to pull and push. If they do not perceive any movement for period of time, they change the orientation of their body and try again. If even this does not work, they detach, re-attach at a different point and try again.

Berman et al. (2011b) studied the same behavior observing how ants interact with fabricated elastic structures. Using the data retrieved from these observations, the authors developed a mathematical model of how collective transport is performed by ants.

In swarm robotics, collective transport behaviors are obtained by using probabilistic finite state machines or artificial evolution. Cooperation is obtained either through explicit communication of the desired motion direction, or through indirect communication, that is, by measuring the force applied to the carried object by the other robots.

Donald et al. (1997) proposed three behaviors based respectively on: force sensing, position sensing and orientation sensing. This work was one of the first works aimed at studying collective transport without a centralized controller and with limited communication.

Campo et al. (2006) proposed a collective behavior in which the robots decide a common direction at the beginning of the experiment by communicating their individual direction. In this way the robots are able to drag an object towards a goal area, even if this area is not perceived by all robots.

Groß and Dorigo (2009) used artificial evolution to tune the parameters of a neural network to achieve collective transport. The obtained behavior was able to cope with different object sizes and weights as well as with different numbers of robots (from 4 to 16). Various metrics and extensive simulations were used to validate the results. In their work, Groß and Dorigo were able to obtain three different transport strategies. In the first, the robot directly connects to the object and move it. In the second, the robots connect to each other and then to the object to move it. In the third, and last strategy the robots are not directly connected to the object but form a circle around the object and push it.

Baldassarre et al. (2006) used artificial evolution and neural networks to perform collective transport. The obtained behavior exploits a sensor able to perceive the force applied by other robots on the chassis. With this sensor, the robots are able to perform collective obstacle avoidance while going towards a target area.

In Ferrante et al. (2013a), we developed a collective transport behavior in which, through communication, a group of robots can agree on a common moving direction towards a goal by averaging the individual desired direction. The proposed solution is able to make robots move towards a common goal while avoiding obstacles. This work was developed for the Swarmanoid project (Dorigo et al., 2012). In Section 4.2 of this dissertation, we present a more detailed analysis of this work using model checking.

Collective decision-making

Collective decision-making deals with how robots influence each other when making choices. It can be used to answer two opposite needs: agreement and specialization. A typical example of agreement in robot swarms is consensus achievement. The desired outcome of consensus achievement is that all the robots of the swarm eventually converge towards a single decision among the possible alternatives. A typical example of specialization, instead, is task allocation. The desired outcome of task allocation is that the robots distribute themselves over the different possible tasks in order to maximize the performance of the swarm.

Consensus achievement. *Consensus achievement* is a collective behavior used to allow a swarm of robots to reach consensus on one choice among different alternatives. The choice is usually the one that maximize the performance of the swarm. Consensus is generally difficult to achieve in swarm of robots because very often the best choice may change over time or may not be evident to the robots due to their limited sensing capabilities.

Consensus achievement is displayed in many insect species. For example, ants are able to decide between the shortest of two paths using pheromones (Camazine et al., 2001). Bees have mechanisms to collectively decide which is the best foraging area or which is the best nest location among several possibilities (Couzin et al., 2005). These mechanisms work even if not all the individuals in the swarm have an opinion on the best choice. Cockroaches also display consensus achievement behaviors when performing aggregation (Amé et al., 2006).

In swarm robotics, the approaches used for consensus achievement can be divided into two categories according to how communication is used. In the first category, direct communication is used: each robot is able to communicate its preferred choice or some related information. In the second category, instead, indirect communication is used: the decision is performed through some indirect clues, such as the density of the robot population.

Wessnitzer and Melhuish (2003) proposed a collective behavior in which robots "hunt" two moving targets. The robots decide which target to follow first, follow it and block it. They then do the same for the second target. Two consensus achievement behaviors are proposed: in the first, the robots simply follow the robot closest to a target, resulting in a decision based on the spatial distribution of the swarm; in the second, the robots vote, using a majority rule, to decide which target to follow.

Garnier et al. (2005, 2009) studied consensus achievement in cockroaches by using a swarm of robots to replicate the experiment by Amé et al. (2006). In their work, consensus achievement is obtained through indirect communication. The focus of this work is both on consensus achievement and aggregation. A mathematical model of the same behavior was developed by Correll and Martinoli (2007). In a similar work, Campo et al. (2011) presented a collective behavior in which the swarm aggregates on the smallest resource that can host the whole group. A further extension was proposed by Francesca et al. (2012). The authors used evolutionary robotics to replicate the results obtained by Amé et al. (2006) comparing the two works also using a macroscopic model.

Gutiérrez et al. (2010) developed a strategy for consensus achievement through direct communication in a swarm of robots performing foraging. The robots are able to decide between two foraging areas. When two robots get close, they exchange their measured distances between the nest and the latest visited goal. Each robot performs an average of its measured distance with the one received from the other robots. In this way, the robots are able to agree on which area is the closest to the nest and discard the other one even when the measured distances are noisy.

Parker and Zhang (2011) proposed a consensus achievement behavior based on quorum sensing. The behavior is inspired by how ants and bees choose the best nest over multiple alternatives (Couzin et al., 2005). When a robot finds a new alternative, it evaluates its quality and sends recruiting messages to other robots to advertise it. The frequency of these messages is proportional to the perceived quality of the alternative. Thanks to the different message frequencies associated with the different alternatives, over time all robots converge on the best alternative. The behavior is implemented as a probabilistic finite state machine.

Montes de Oca et al. (2011) focused on consensus achievement in a scenario where robots perform multiple parallel executions of collective transport in groups of three from a nest area to a goal area. The robots need to reach consensus between two possible paths, one longer than the other. Each individual robot has a preferred path. When a group of three robots is formed in the nest, the robots choose the path that is preferred by the majority of them. The chosen path becomes the preferred one for all the robots in the group. Since the robots choosing the short path take less time to complete the execution, they are more often in the nest. This results in more groups formed by robots preferring the short path than those preferring the long path. This asymmetry eventually makes the robots use the shortest path. In this work, consensus is achieved both through direct and indirect communication, as the robots use direct communication at group level, and indirect communication at the swarm level. In Section 4.1 of this dissertation, we present a more detailed analysis of this work using model checking.

Task allocation. *Task allocation* is a collective behavior in which robots distribute themselves over different tasks. The goal is to maximize the performance of the swarm by letting the robots dynamically choose which task to perform.

Task allocation can be observed in natural systems such as ant and bee colonies (Theraulaz et al., 1998). For example, in ant or bee colonies, part of the swarm can perform foraging while another part looks after the larvae. Task allocation is not fixed but can change over time.

In swarm robotics, task allocation is mainly obtained through the use of probabilistic finite state machines. To promote specialization, the probabilities of selecting one of the available tasks are either different among the robots or they can change in response to task execution or messages from other robots. In swarm robotics, task allocation has been studied mainly on robots performing foraging.

In one of the first works on task allocation, Krieger and Billeter (2000) developed a very simple, threshold based mechanism. Robots have to collect objects that are then converted into energy in the nest. While foraging, the robots consume energy. To replenish this energy, the robots can draw it from a common reservoir. Each robot decides to leave and collect objects or to stay in the nest according to a probability. This probability depends on whether the nest energy is above or below a given threshold. Since this threshold is different for each robot in the swarm, the number of robots allocated to foraging or to resting is a function of the energy level of the nest.

Agassounon and Martinoli (2002) studied task allocation in a foraging task similar to the one studied by Krieger and Billeter (2000). However, in this case the probability to select the foraging task or the resting task depends on individual observations of the environment and of other robots. Thus, the probability is a function of the success or failure of the last foraging trial, of the frequency with which other robots are encountered when foraging or of the perceived density of objects. A mathematical model of a similar task allocation behavior has been developed by Liu et al. (2007).

Yun et al. (2009) studied the problem of how to allocate robots on a construction site so that the number of assembling operation to do is shared equally. Each robot computes optimal equal-mass partitions, that is, partitions with the same number of operations, by sharing information with its neighbors. The developed behavior is robust to changes in the environment and scalable with the number of robots.

Pini et al. (2009) developed a task allocation behavior in a swarm of robots performing foraging using a bucket brigade approach. In this work, the experiment arena is divided in three areas, the first one is the nest, the second one is an exchange area and the third one is where the objects are. In the exchange area, the robots have the possibility to wait for other robots in order to exchange the object in a bucket-brigade fashion. Through different thresholds on the waiting time, the robots autonomously change their role between those who bring objects from the source to the exchange area and those who bring objects from the exchange area to the nest.

Halász et al. (2012) studied task allocation using robots performing stick pulling. Robots must remove sticks scattered in the environment. To remove a stick, two robots must cooperate performing two part of the tasks: one holds a stick from the top and the other from the bottom. Once a robot finds a stick, it holds its top and waits for another robot to complete the second part of the task. If after a certain waiting time no one helped, the robot leaves the stick and searches for another one. This waiting time is changed dynamically according to how well a robot performed in the past. Although the results were not conclusive, the authors observed that, over time, the robots develop a preference for one of the two parts of the tasks, specializing in robots holding the top part of a stick and robots holding the bottom part.

Pini et al. (2011) considered a situation in which robots can choose between carrying an object directly from the source to the nest and storing it in a dedicated two-sided structure called TAM (Brutschy et al., 2012) (see Appendix A.2). Stored objects can be collected by robots waiting on the other side of the structure and carried to the nest. The authors develop a mechanism that allows the robots to choose whether to use the structure on the basis of the cost involved.

Other collective behaviors

In this section, we present some works in swarm robotics that we consider significant but that do not fall in any of the categories presented above.

Collective fault detection. Autonomous robots have still a limited reliability. Even though the quality and robustness of the hardware is increasing, hardware failures are still quite common. Techniques to allow robots to autonomously detect failures and faulty behaviors have been developed.

Christensen et al. (2009) developed a swarm-level fault detection behavior based on firefly synchronization. All the robots in the swarm are emitting a signal in a synchronous way. The robots are able to perceive if another robot is in a faulty state by observing if it is synchronized with them. If a robot is not synchronized, it is assumed to be faulty and a response is initiated.

Group size regulation. Group size regulation is the collective capability of creating or selecting a group of a desired size. This can be useful for many reasons. For example, Lerman and Galstyan (2002) showed that an excessive number of robots can reduce the performance of the swarm, and demonstrated for different behaviors that it is possible to identify a group size that maximizes the performance of the swarm.

Melhuish et al. (1999a) developed a behavior inspired by fireflies to achieve the formation of groups of the desired size. Each robot can emit, at a random time, a signal. The robots then count the number of signals received over a period. The obtained number can be used by the robots to estimate the size of the group and thus to create groups of the desired size. In a related work (Brambilla et al., 2009), we improved the original behavior by introducing a more strict signaling order. With this improvement we were able to obtain a more robust and reliable estimate of the size of the group.

Pinciroli et al. (2013) studied a collective behavior able to form groups of robots of the desired size. The swarm is composed of flying robots and ground robots. The ground robots perform aggregation under the flying robots. The probabilities used by them to join or leave a group are communicated by the flying robots according to the size of the group itself. With this simple mechanism the robots are able to form groups of various sizes.

Human-swarm interaction. Robot swarms are conceived to be autonomous and to make decisions in a distributed way. While these are in general considered to be positive features, they also limit the degree of control of a human operator over the swarm. In fact, since there is neither a leader nor centralized control, the operator does not have a simple way to control the behavior of the swarm. Human-swarm interaction studies how a human operator can control a swarm and receive feedback information from it.

McLurkin et al. (2006) developed a simple mechanism in which robots are able to provide information to human operators using LEDs and sound.

Naghsh et al. (2008) proposed an analysis of different possible approaches to human-swarm interaction, classifying them in direct human-swarm interaction, direct swarm-human interaction, and remote interaction via base station.

Podevijn et al. (2012) used a Microsoft Kinect system to give commands through gestures to a swarm of robots. The human operator is able to command the swarm to select, split, merge and rotate.

Giusti et al. (2012) used a similar approach based on gestures. The robots observe the gestures of a human operator. Each robot is able to guess the performed gesture, but due to its limited vision capabilities, different robots could make a different guess. To reach consensus, the robots vote using multi-hop communication and finally execute the order associated with the performed gesture.

Kolling et al. (2012) presented two approaches to control a swarm. The first approach is based on global communication: a human operator uses a central computer to select and control a subgroup of robots. The second approach is based on local interactions: the human operator places pre-programmed beacons in the environment. These beacons are used to communicate a new behavior to the robots that are in their communication range.

2.1.4 Open issues in swarm robotics

Swarm robotics has several possible applications, including: exploration, surveillance, search and rescue, humanitarian demining, intrusion tracking, cleaning, inspection and transportation of large objects. Despite their potential to be robust, scalable and flexible, up to now, robot swarms have never been used to tackle a real-world application and are still confined to the world of academic research. At the current state of development of the swarm robotics field, the focus is mostly on obtaining a desired collective behaviors and understanding their properties. For this reason, researchers usually tackle simplified testbed application, such as foraging and construction.

Foraging is the most used testbed application in swarm robotics. Robots have to retrieve "prey" objects from an environment and bring them back to a "nest". Foraging, while simple, can be considered as an abstraction of more complex applications, such as demining and search and rescue. Foraging is also used to investigate the effect of interference in robot swarms (Lerman and Galstyan, 2002). In particular, foraging is commonly used as a testbed for collective exploration, collective transport and collective decision-making.

Another testbed application that has attracted a lot of interest recently is construction. Swarms of robots could be used to build complex structures in those cases in which humans would be unable to, such as underwater and in space. Moreover, construction could greatly benefit from the parallelism and flexibility of robot swarms. Construction is a complex task that requires the combination of several collective behaviors, such as object clustering and assembling to gather material, collective transport to carry material, and collective decision-making to allocate the robots to the different sub-tasks of the construction process. A recent example of construction performed by flying robots can be found in the work by Lindsey et al. (2012). Even though this construction system cannot be considered a swarm, as the robots exploit a centralized system for localization and action planning, the presented problem can be considered as an interesting testbed for swarm robotics.

There are many possible reasons for the absence of robot swarms in the real world, such as the hardware limitations of the available robots. We foresee that, in the near future, swarm robotics will be used more and more frequently to tackle real-world applications. With an increasing use of robot swarms, we envision an increasing need for a swarm engineering, that is, a need for methods for: 1) requirement modeling and specification, 2) design and realization, 3) verification and validation, and 4) operation and maintenance. In the following, we analyze how these aspects of swarm engineering have been tackled and we discuss some open problems.

Requirement modeling and specification. With the application of swarm robotics to real-world scenarios, we foresee an increasing need for well defined processes to help in requirement gathering and for formal languages to help in requirement specification. Up to now, none of these processes have been studied directly in swarm robotics. This is probably due to the lack of real-world applications. Effort will therefore be necessary to understand whether existing requirement gathering processes and existing requirement specification languages from other fields can be re-used or adapted to swarm robotics or whether new ones need to be developed.

Property-driven design, the design approach proposed in this dissertation, can be considered a first approach to the requirement specification problem. More details can be found in Chapter 3.

Design and realization. The design aspect of swarm engineering was discussed thoroughly in Section 2.1.1. One main issue remains open: the lack of general and effective methods for the top-down design of collective behaviors. Automatic design methods can be considered as top-down methods. However, even if these methods are improving, a lot of domain knowledge is still required to tackle medium to complex applications, in particular to define the setup of the automatic process.

In this dissertation, we present a novel approach for the top-down design of robot swarms based on prescriptive modeling and model checking. For more details, see Chapter 3.

Verification and validation. Verification and validation exploit analysis methods, which have been discussed in detail in Section 2.1.2. Despite the great number of analysis methods, performing verification and validation of a robot swarm and comparing one system with another is still very difficult. The reason behind this is the lack of well defined metrics and testbed applications. Very often, metrics are too tightly related to a specific solution and thus cannot be reused for other systems or for comparisons. The lack of common metrics is also related to the lack of well defined testbed applications. As said, foraging and construction are the only commonly used testbed applications. However, foraging is limited in its use to some collective behaviors, such as task allocation or area coverage and cannot be used in others. Moreover, as discussed in the taxonomy presented by Winfield (2009), there is no single definition of a standard foraging scenario. In order to promote the comparison of robot swarms, it would be necessary to define a set of standard foraging scenarios and promote the distribution of open-source behaviors and public available datasets. Construction, as a testbed, suffers from similar limitations.

In addition, in order to apply swarm robotics to real-world applications it will be necessary to formally verify properties of robot swarms, such as their safety. In this dissertation, we show how formal methods can be used to perform verification. More details can be found in Chapter 4.

Operation and maintenance. Robot swarms have the potential advantage to require limited manual intervention because of their fault tolerance, scalability and flexibility. Although these three characteristics might reduce the need for maintenance, this might be true only up to a given extent. Further studies are necessary to understand when and how to perform maintenance on a robot swarm. Moreover it is necessary to study if it is possible to derive general maintenance principles or if different collective behaviors need different maintenance approaches.

Regarding operation, one key issue is how to let humans and swarms cooperate. In fact, due to the lack of a centralized controller, it is in general very difficult to effectively control a swarm once it starts operating. This means that, for example, it might be difficult to stop a swarm that is behaving in an unpredicted or dangerous way. Some studies on human-swarm interaction have been recently published (see Section 2.1.3) but this issue still remains open.

2.2 Formal methods

The year 1999 marked the first use of artificial intelligence in a space mission: the *Remote Agent*, NASA's autonomous spacecraft controller, was used to pilot the Deep Space 1 flight (Rayman et al., 2000). NASA's Remote Agent was a complex piece of software that strongly exploited concurrency. To ensure its reliability, before being employed, it was subject to more than 300 hours of testing. Unfortunately, despite this in depth testing, a deadlock occurred, putting at risk the success of the mission. The problem was soon identified and solved. The mission was completed with success, but the need for a more formal and complete verification of the reliability of software was once more highlighted.

In the aftermath of the problem, Havelund et al. (2001) performed a complete analysis of Remote Agent's software to search for possible deadlocks. They did not perform the analysis manually. Instead, they used *model checking*, a newly developed technique at that time. Thanks to model checking, the authors were able to identify five potential deadlock conditions in the software, among which the one that happened during the real mission.

Model checking (Clarke, 1997) is a technique that can be used to analyze a system: given a model of the system, using model checking it is possible to formally prove that the model satisfies a given property, such as the absence of deadlocks. It can be used in safe-critical applications in which simulations and experiments might not be enough to guarantee the correctness of a system. In fact, simulations and experiments can only test a subset of all possible execution scenarios of a system. Model checking, instead, perform an automatic and exhaustive analysis of the model to formally verify whether the given properties are satisfied by the given model. This approach has been applied with success to several different fields, such as electronic circuits (Burch et al., 1990), space-craft controllers (Havelund et al., 2001), embedded real-time systems (Stankovic, 1996), wireless sensor networks (Clarke, 1997) and mobile robotics (Jeyaraman et al., 2005, Fisher and Wooldridge, 1997, Bruni et al., 2012). It was recently applied also to the verification of robot swarms (Konur et al., 2012).

In its simplest form, model checking requires two components: a *model* of the system to analyze and a list of *properties* that the system must satisfy. Model checking is then performed using *model checkers*.

In this section, we present the formalisms and languages used to perform model checking for the design and analysis of robot swarms.

There are several formalisms that can be used to define a model for model checking (Berard et al., 2010). In this dissertation, we use two families of formalisms: Markov chains and process algebras. Even though some differences exists between Markov chains and process algebras, these two families of formalisms are strongly connected, to the point that it is possible to formally prove a structural correspondence between the two (Brinksma and Hermanns, 2001). In Section 2.2.1, we present the general concepts behind Markov chains and process algebras. We postpone the presentation of the details of the two process algebras used in this dissertation, Bio-PEPA and Klaim, to Chapter 4.

There are also several languages that can be used to specify properties for model checking (Berard et al., 2010). All languages used to define properties for model checking are based on mathematical logic. In this dissertation, we use three different logics: probabilistic computation time logics (PCTL), continuous stochastic logic (CSL) and mobile stochastic logic (MoSL). All these three logics belong to the family of stochastic temporal logics. Stochastic temporal logics will be presented in Section 2.2.2. We postpone the presentation of the details of each particular logic used in this dissertation to Chapter 4.

Finally, after presenting the formalism to define models and the logics to specify properties, in Section 2.2.3, we present the basic concepts of model checking and statistical model checking.

2.2.1 Model formalisms for model checking

In this section, we present the two families of formalisms used to model robot swarms for model checking: Markov chains and process algebras. In particular, we briefly present how Markov chains can be used for modeling a robot swarm. For a complete review of modeling and analysis techniques in swarm robotics see Section 2.1.2. For a complete and formal presentation of Markov chains see Norris (1998). We also give a brief description of the characteristics common to all process algebras. More informations on process algebras can be found in Fokkink (2000). More information about the two process algebras used in this dissertation, Bio-PEPA and Klaim, can be found in Chapter 4. A discussion about the relationship between process algebras and Markov chains can be found in Brinksma and Hermanns (2001).

Markov Chains

Markov chains can be used to model or design the behavior of the robots: *states* represent actions that robots can perform, such as `random walk` and `grasp` `object`. Transitions link two states and are activated through *transition conditions* such as `obstacle seen` and `object grasped`. As an example of a Markov chain of a robot performing foraging, see Figure 2.2a. Markov chains must have a starting node and a finite or countable number of states.

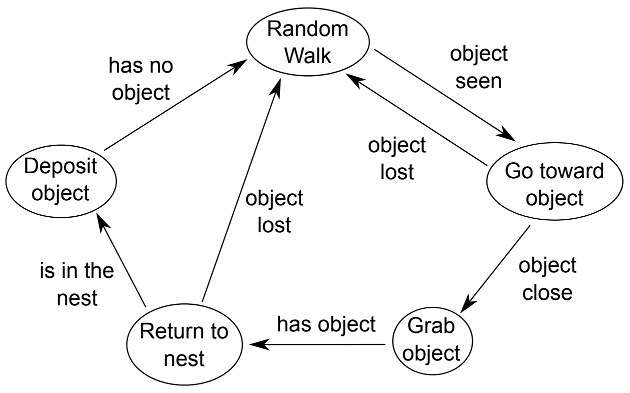
Markov chains can be used to model a robot swarm in two ways: as a *microscopic Markov chain*, that is, a model that consider each individual robots, and as a *macroscopic Markov chain*, that is, a model that consider the swarm as a whole.

A microscopic Markov chain describes the behavior of the individual robots and their interactions. In a microscopic Markov chain, the collective behavior of the swarm is usually developed starting from the *and*-composition of the Markov chains describing the individual behaviors of the robots (Dixon et al., 2011). Figure 2.2b shows an example of a microscopic Markov chain of a swarm of four foraging robots.

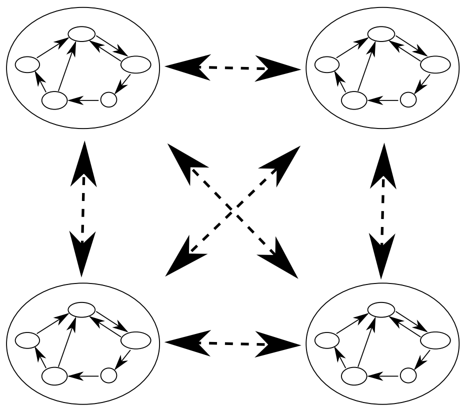
A macroscopic Markov chain, instead, describes the swarm as a whole, without considering the individual robots composing it. In general, similar to models based on rate equations, a macroscopic Markov chain is developed starting from a Markov chain describing the behavior of a generic individual of the swarm augmented by associating a counter to each state. Such counters are used to keep track of the number of robots that are in the associated state. It is important to note that, differently from rate equation models, the counter is not the proportion of the total number of robots in the associated state, but the effective number of robots. Figure 2.2c shows an example of a macroscopic Markov chain of a swarm of foraging robots.

When compared to macroscopic Markov chains, microscopic Markov chains give a finer description of the robots and their interactions, which are fundamental components of any robot swarm. However, they suffer from the *state-space explosion* problem (Pelánek, 2009): in a microscopic Markov chain, the number of states grows exponential with the number of robots. The number of states in a microscopic Markov chain is k^n, where n is the number of robots and k is the number of states of the Markov chain of each individual robot. In a macroscopic Markov chain instead, the number of states follows the binomial coefficient $\binom{n+k-1}{k-1}$.

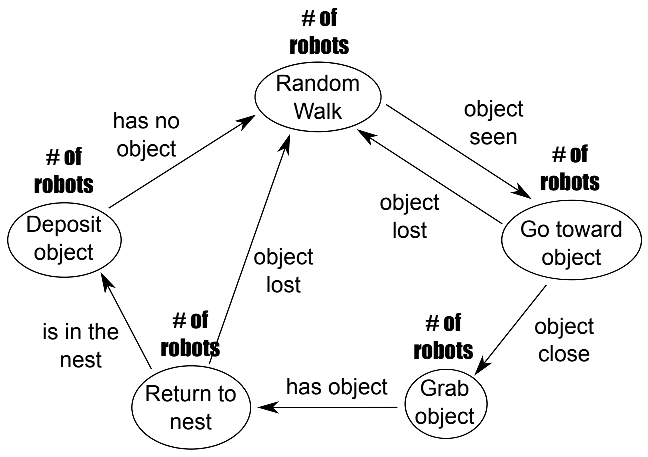
As it is possible to see in Table 2.1, even for very small swarms, the number of states composing a microscopic Markov chain gets very large, making microscopic Markov chains computationally intractable using standard techniques. For this reason, macroscopic Markov chains are usually employed to model robot



(a) A Markov chain describing the behavior of an individual foraging robot.



(b) A simplified view of a microscopic Markov chain of a foraging swarm composed of four robots. The model is the *and*-composition of the Markov chains describing the behavior of the individual robots, as in Figure 2.2a.



(c) A simplified view of a macroscopic Markov chain of a foraging swarm. The model consists of the Markov chain describing the behavior of the individual robots, as in Figure 2.2a, to which counters are associated.

Figure 2.2: Different approaches to modeling a robot swarm using Markov chains.

Table 2.1: The number of states of a microscopic and macroscopic Markov chain as the swarm size increases. We consider a swarm in which the behavior of the individual robots is composed of $k = 5$ states, as the one in Figure 2.2a.

Number of robots	Macro	Micro
n	$\binom{n+k-1}{k-1}$	k^n
1	5	5
5	126	3125
10	1001	$9.76 * 10^6$
50	$3.16 * 10^5$	$8.88 * 10^{34}$
100	$4.60 * 10^6$	$7.89 * 10^{69}$
500	$2.66 * 10^9$	$3.05 * 10^{349}$
1000	$4.21 * 10^{10}$	$9.33 * 10^{698}$

swarms (Brambilla et al., 2013).

In this dissertation, we used a macroscopic Markov chain to analyze a collective decision-making behavior, see Section 4.1, and a microscopic model to analyze a collective transport behavior, see Section 4.2.

Time in Markov chains can be modeled in two different ways: discrete and continuous. In discrete time Markov chains (DTMC), time assumes only values in $\mathbb{Z}^+$, whereas in continuous time Markov chains (CTMC), time can assume any value in $\mathbb{R}^+$. The choice between DTMC and CTMC depends on the system to model: in case time is not a critical aspect and can be easily discretized, DTMC are more convenient; on the contrary, when it is important to keep precisely track of times, DTMC should be preferred. Note that DTMCs and CTMCs have the same expressive power (Serfozo, 1979).

Practically, one of the main difference between DTMC and CTMC lies in the meaning of the transition parameters. In DTMC, transition parameters represent the probability p that a robot moves from a state to another over a fixed time period; for this reason, transition parameters for DTMC must be in the interval $[0, 1]$. In CTMC, instead, transitions parameters represents the rate λ at which a robot moves from a state to another. This rate follows an exponential distribution of parameter $\lambda \in (0, \infty)$. Restriction to exponential distributions does not represent a real limitation since it possible to model most random distribution by suitable combinations of negative exponential ones (Fang, 2001).

A last kind of Markov chains are Markov chains with *reward structures*. Reward structures are real valued quantities that can be assigned to states or transitions. They can be used to reason about the expected value of the amount of times a certain transition happens or a certain state is reached.

Process algebras

A process algebra (Bergstra and Klop, 1984, Bergstra et al., 2001), also called process calculus, is a formal language used to specify and describe distributed systems. In particular, process algebras are well suited for modeling systems characterized by communication and concurrency. Distributed systems are described using process algebras as collections of *processes* and *actions* they can execute. Interactions between processes is described in terms of communication.

Process algebras are characterized by having a formally defined *structured operational semantics* that structures processes into *labelled transition systems*, that is, a series of states and transitions that describe how the system evolves. Drawing a parallel with Markov chains, in process algebras, we can interpret processes as states and actions as transitions: for example, $P \xrightarrow{a} Q$ means that process P performs the atomic action a after which it behaves as Q. This can be used to model robots that perform differently after completing a specific action. For example, in modeling a search and retrieve scenario, a robot searching for an object may perform the action `find` after which it behaves as a retrieving robot: $R\,searching \xrightarrow{find} R\,retrieving$. By defining processes, actions and their relationship it is possible to describe a dynamic system.

One of the characteristics of process algebras is the ability to define processes hierarchically, that is, the ability to define a process as a composition of concurrent components. In this way, a "robot swarm process" can be seen as composed of several "individual robot processes". There are several ways to compose processes: the most important two are parallel composition and sequential composition. Many other compositional operators are available, we refer the interested reader to Fokkink (2000).

Prescriptive and descriptive models

In this dissertation, we employ models both for the design (see Chapter 3) and analysis (see Chapter 4) of robot swarms. Models used for the design and analysis of robot swarms are technically identically: the same formalisms and the same modeling techniques are used to develop them. However, they are conceptually different. In fact, the models used for the design of robot swarms describe systems that have not been developed yet, that is, they are used to develop a system, not to describe it. We call this kind of models *prescriptive* *models*. Instead, the models for the analysis of robot swarms describe systems that are already completed. We call this kind of models *descriptive models*. In this dissertation, we use both prescriptive and descriptive models.

The difference between prescriptive and descriptive models is not only semantic. In fact, there is a fundamental difference between these two kinds of models: how parameters are set. In descriptive modeling, the parameters of the model are derived from the observations of the system being described. Usually, these parameters are chosen so that the analysis of the model produces results that are comparable with those of the real system. In prescriptive models, this is not possible, as there is no system from which one can derive the parameters. Parameters of prescriptive models are thus derived from the informations available before the development of the system, such as the geometry of the environment and through educated guess.

2.2.2 Property definition languages for model checking

The most common way to formally express properties in model checking is through the use of logic predicates (Clarke, 1997). In this dissertation, we employ probabilistic temporal logics. Probabilistic temporal logics are a family of mathematical logics that allow us to express concepts related to time in which propositions can be, not only true or false, but also have a certain probability of being true. Probabilistic temporal logics are well suited for swarm robotics as they can capture both the time-related and stochastic aspects that characterize robot swarms.

Among the many mathematical logics belonging to the family of probabilistic temporal logic, in this dissertation, we use probabilistic computation time logics (PCTL), continuous stochastic logic (CSL) and mobile stochastic logic (MoSL). In this section, we limit our discussion to PCTL as it is the simplest of the three. We do not discuss in details CSL, as it can be considered, for the goals and purposes of this dissertation, a continuous-time version of PCTL. Some relevant parts of CSL will be presented in Section 4.1. MoSL, an extension of CSL, will be presented in Section 4.2.

PCTL, originally developed by Hansson and Jonsson (1994) is a probabilistic extension of CTL (Computation Tree Logic), a branching time logic. CTL and PCTL are based on the concept of computation tree. A computation tree can be used to represent the temporal evolution of a Markov chain. It consists of a potentially infinite rooted tree in which the root is the initial state of a

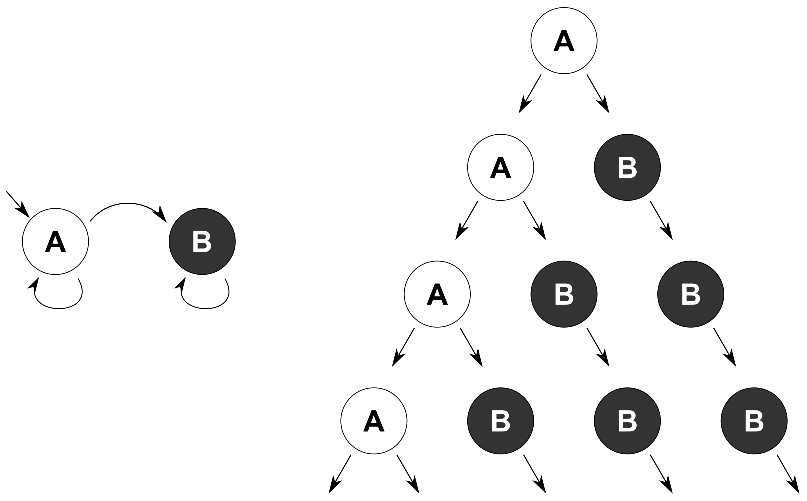


Figure 2.3: A simple Markov chain (on the left) and part of its computation tree (on the right).

corresponding Markov chain, and each node is a possible state of the system. Edges link a state with its next possible states. Each path on the tree represents a possible execution of the system. Since a sequence of nodes represents the time evolution of a system, the transition between two nodes is usually called a time-step. An example of a simple Markov chain and its computation tree is displayed in Figure 2.3.

On a computation tree it is possible to analyze temporal properties due to the fact that the computation tree gives us a temporal evolution of the Markov chain. Examples of such properties are: `eventually the system will reach state B` or `if the system starts from state B then it will never reach state A`. These properties can be expressed using computation tree logic (CTL).

PCTL extends CTL by introducing probabilities. It is thus possible to express properties such as *property α will eventually become true with probability 0.45* or *there is a 0.7 probability that α will hold true for 10 seconds*. Understanding the details of PCTL is not essential to understand the general principles of this dissertation. Nonetheless, we think that a simple introduction (based on Hansson and Jonsson (1994) and Ciesinski and Größer (2004)) can be useful for those interested in using PCTL for model checking in swarm robotics.

A brief introduction to PCTL

The syntax of PCTL is composed of *state formulae*, generally identified by the upper case greek letter Φ, and *path formulae*, generally identified by the lower case greek letter ϕ. Path formulae are infinite sequences of state formulae ordered over time: $\phi_t = \Phi_0 \rightarrow \Phi_1 \rightarrow \ldots \rightarrow \Phi_i \rightarrow \ldots$, where ϕ_t is a generic path. State and path formulae allow us to define properties both on events that are related to a single state, such as the probability of a deadlock, and events that span over multiple states, such as the ability of a system to complete a task after recovering from a failure.

Standard logic constants and operators, such as $\top$, $\wedge$, $\Rightarrow$, $\neg$, are available in PCTL together with operators for probabilistic and temporal properties. Formulae like $\forall\,\phi$ and $\exists\,\phi$ are replaced by $P_{\bowtie p}[\phi]$, where P indicates the probability operator, $p \in [0,1] \subset \mathbb{R}$ is a probability limit and $\bowtie$ is a placeholder for $\{>, \geq, \leq, <\}$. An example of such formulae is $P_{\geq p}(\phi)$, which asserts that the probability that ϕ holds true is at least p.

Two temporal operators are available in PCTL: X, called *next* and $\mathcal{U}^{\leq t}$, called *bounded until*. $X\phi$ denotes that ϕ holds in the next state. The bounded until operator deserves a more in depth analysis, since it is less immediate than the next operator, and it constitutes the basis for deriving more complex temporal operators.

$\phi_1 \mathcal{U}^{\leq t} \phi_2$ denotes that ϕ_1 has to hold from now until, within at most t time units, ϕ_2 becomes true. More formally, considering σ as a path of the model,

$$\sigma \models \phi_1 U^{\leq t} \phi_2 \Leftrightarrow \exists\, t_1 \leq t.\sigma(t_1) \models \phi_2 \wedge \forall\, t_0 \leq t_1.\sigma(t_0) \models \phi_1$$

were $\sigma(t)$ is the state in σ occupied at time t and $\sigma \models \phi$ means that path σ satisfies formula ϕ. In the variant without time bound t, it is required that eventually a state is reached in σ in which ϕ_2 holds, and that all preceding states satisfy ϕ_1. Again, more formally:

$$\sigma \models \phi_1 U \phi_2 \Leftrightarrow \exists\, t_1.\sigma(t_1) \models \phi_2 \wedge \forall\, t_0 \leq t_1.\sigma(t_0) \models \phi_1$$

The next and until operators are the only necessary operators to express temporal conditions. Other useful temporal operators can be derived from these two. Of particular interest are: $\diamond$, which reads as "eventually"; $\diamond^{\leq t}$, which reads as "sometimes within the next t steps"; $\square$, which reads as "always"; and $\square^{\leq t}$, which reads as "always within the next t steps".

In case the Markov chain being analyzed is augmented with rewards structures, it is possible to express properties about them by using *reward formulae*. We consider two types of reward formulae: $R\{rwlabel\}_{\bowtie p}[F\ proposition]$ and $R\{rwlabel\}_{\bowtie p}[C^{\leq t}]$, where *rwlabel* is the name of the *reward structure* in the model

Table 2.2: Some examples of PCTL formulae with their natural language equivalent.

Formula	Natural language equivalent
$P_{\geq 0.75}[\diamond^{\leq 1000} task\_completed]$	With a probability greater than 0.75 the system completes the task before 1000 time-steps.
$P_{\leq 0.05}[\square waiting\_robots > 10]$	The probability that there will be more than 10 robots blocked in the waiting states is less than 0.05. This means that there could be more than 10 resting robots at a certain point in time, but some of them will be eventually start working again.
$P_{>0.95}[\square(request \rightarrow \diamond^{\leq 10} response)]$	There is a probability greater than 0.95 that every request is answered within 10 time-steps.
$P_{<0.10}[\diamond\square deadlock]$	The probability that the system will eventually get in a deadlock is less than 0.10.

which the formula refers to, $\bowtie$ is a placeholder for $\{>, \geq, \leq, <\}$, $p \in [0,1]$ is a probability, F returns the expected value of a reward structure accumulated until a state is reached where *proposition* holds, and C returns the expected value of the reward structure up to time t. Note that, in reward formulae, $\bowtie p$ can be replaced by $=?$, which returns the expected probability value instead of performing a comparison with a predefined value.

Together, probabilistic and temporal operators make PCTL a very flexible and powerful logic, which can be used to express many interesting properties of particular interest for swarm robotics. The use of probabilistic temporal logics allows the developer to express properties that are difficult or impossible to express using algebraic mathematics.

In Table 2.2, we list some examples of PCTL formulae with the double intent of demonstrating the capabilities of PCTL and of listing some common properties of robot swarms.

2.2.3 Model checking

Having presented model formalisms and logics to define properties, we now have all the elements necessary to perform model checking. Model checking can be used to verify, over all possible executions, that a system satisfies a set of properties. This capability is an advantage over simulations as the probability to find a problem in the system using model checking is not related to the probability that the problem itself manifests.

In its simplest form, model checking is performed through *state space enumeration*: all possible executions of a model are enumerated and considered one by one to verify whether the desired property are valid or not. Practically, this means that the computation tree of a model (see Figure 2.3) is explored depth-first, and a property is verified on each resulting path. More advanced techniques can be employed in order to avoid the complete exploration of the computation tree. The current state-of-the-art approaches are based on binary decision diagrams (Kwiatkowska et al., 2004).

Model checking allows the user not only to verify that a model satisfies a specific probabilistic property (e.g., $P_{\geq 0.75}[\phi] ?\, \texttt{TRUE}$), but also to compute with which probability the model satisfies it ($P_{=?}[\phi] ?\, 0.86$). This characteristic is very useful to find the best parameters of a model that maximize the probability to satisfy a specific property.

Using model checking, it is also possible to analyze the probability distribution of a property, for instance to compute the variance of the observed variable. This would be impossible with fluid flow analysis, as using fluid flow analysis it is only possible to obtain the expected value of an observed variable. Additionally, model checking can be used also as a diagnostic tool, as it provides counterexamples for non-validated properties, which can help in the debug phase.

A first limit of model checking is that it verifies a system model, and not the system itself: for this reason the quality of the results obtained with model checking depends on the quality of the model used. In general thus, model checking is a good approach to validate the design of a system, not its implementation. Complementary techniques, such as testing or model validation, are necessary to find implementation problems.

The main limit of model checking is that, in general, it is impossible to analyze models composed of a high number of states. This problem, known as “state explosion”, has been subject of many studies and techniques to reduce the number of states of a model have been developed (Pelánek, 2009). Despite such techniques, even state-of-the-art model checkers cannot handle models larger than 10^{10} states (Kwiatkowska et al., 2004).

As said in Section 2.2.1, unfortunately, simple microscopic Markov chains used to model robot swarms can rapidly become larger than 10^{10} when the size of the swarm increases. For this reason, model checking in swarm robotics has been performed usually using macroscopic Markov chains with a relatively limited number of robots.

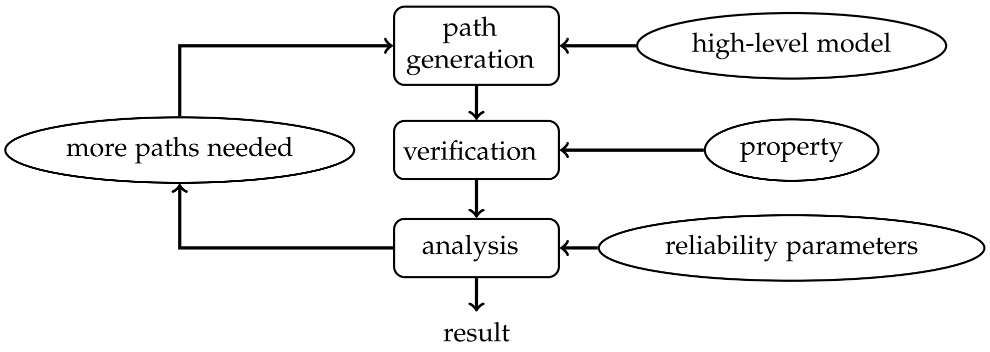


Figure 2.4: Overview of the statistical model checking approach

A way to overcome this problem is *statistical model checking*. Statistical model checking, also known as approximate model checking, is a novel approach to model checking (Nimal, 2010). Compared to traditional model checking, statistical model checking does not explore completely the state space of a model. Instead, it samples a large but limited number of executions of the model and uses statistic estimators to compute the result.

Several solving techniques are available for statistical model checking (Nimal, 2010). The most used is the *confidence interval* method. An overview of the statistical model checking approach is given in Figure 2.4. In statistical model checking, confidence interval methods provide an estimate of the probability with which a given property holds with a certain level of reliability. A confidence interval is an estimated interval of a certain width $2w$ such that, if the estimation is repeated a number of times, then the real probability lays within this interval $100 \times (1 - \alpha)\%$ of the times. The reliability parameter α is the level of confidence. Assume, for all $i \in \{1, \dots, N\}$, that $\{Y_i\}_i$ is a set of realizations of the Bernoulli random variables X_i, where X_i is 1 if property ϕ on a randomly generated path σ of length k holds, and 0 otherwise. It is assumed that all Y_i are independent and identically distributed (i.i.d.) and normally distributed. Using the central-limit theorem it is possible to derive a lower bound on the required number of paths N that need to be generated in order to provide an estimate of the probability with the required accuracy w and level of confidence α. It is also possible, given α and a desired number of paths N, to calculate the accuracy w. Several other methods are available as well, such as the asymptotic confidence interval method (ACI) and the approximate model checking technique (AMC) that use different bounds for the minimal sample size N. The latter also uses different notions of accuracy and confidence. For a detailed comparison of these methods we refer to the work by Nimal (2010).

The confidence interval method has also been adapted to estimate the expected value of rewards, that is, for *reward formulae* of type $R_{=?}[\phi]$. Let Σ be a *reward structure* and ϕ a property over paths σ. The random variable $X_{\phi,\Sigma}(\sigma)$ can now be defined to produce a reward value, that is, it is of type $X_{\phi,\Sigma}(\sigma) \in \Omega \to \mathbb{R}^+$. It is assumed that the random variables are i.i.d. and normally distributed. For the rest, the method is similar to the confidence interval method described above.

Using statistical model checking it is possible to perform model checking also on very large models, even on microscopic models of robot swarms. In this dissertation, we employ both complete model checking and statistical model checking.

Chapter 3

Design of robot swarms using model checking

3.1 Introduction

In this section, we present property-driven design, a top-down design method for robot swarms based on prescriptive modeling and model checking. The developer creates a prescriptive model of the desired robot swarm and uses it as a blueprint for the implementation and improvement of the final swarm. The use of model checking allows the developer to formally verify properties directly on the model, reducing the need for testing in simulation or with robots. In property-driven design, different "views" of the system to realize are produced, from the most abstract (the properties of the system) to the most concrete (the final robot swarm). This is similar to model-driven engineering (Miller and Mukerji, 2003) where software is designed through a series of model transformations from platform-independent models to executable platform-specific models.

Property-driven design addresses the shortcomings of the existing approaches.

- It aims at providing a method to formally specify the requirements of the desired robot swarm;
- It reduces the risk of developing the "wrong" robot swarm, that is, a robot swarm that does not satisfy the requirements;
- It promotes the re-use of available models and tested solutions;
- It can be used to develop platform-independent models that help in identifying the best robotic platform to use;

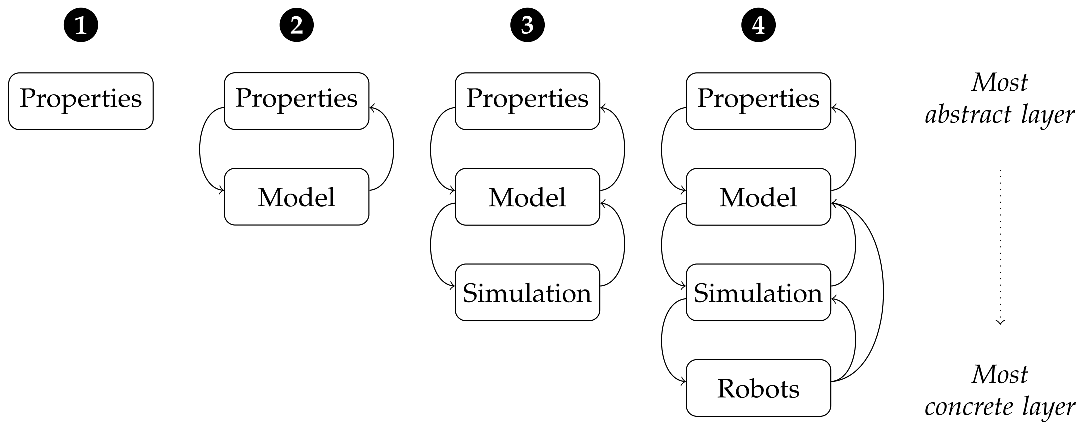


Figure 3.1: The four phases of property-driven design.

- It helps to shift the focus of the development process from implementation to design.

Property-driven design is a step forward in the development of *swarm engineering*: the systematic application of scientific and technical knowledge to specify requirements, design, realize, verify, validate, operate and maintain an artificial swarm intelligence system (Brambilla et al., 2013).

To illustrate and validate property-driven design, we apply it to two case studies: aggregation and foraging.

In Section 3.2, we present property-driven design. In Section 3.3, we present the two case studies. In Section 3.3.3, we discuss the obtained results.

This chapter is based on Brambilla et al. (2012, 2014a)

3.2 Property driven design

Property-driven design is composed of four-phases: i) the requirements of the robot swarm are first formally described in the form of desired properties; ii) subsequently, a prescriptive model of the robot swarm is created; iii) this prescriptive model is used as a blueprint to implement and improve a simulated version of the desired robot swarm; iv) the final robot swarm is implemented.

A schema showing the different phases of property-driven design is presented in Figure 3.1.

In each of the phases of property-driven design, a new layer is added to the system. Layers differ in their level of abstraction: the *properties* layer is the most abstract, in which only the goal characteristics of the robot swarm are stated; the *robots* layer is the most concrete, in which the actual software for the real robots is developed and deployed. The addition of a new layer brings the system closer to its final state.

Each phase of property-driven design is characterized by a *development/validation* cycle: the focus of the developer is on the newly introduced layer, but all previously developed layers are still active, that is, they are still improved and expanded, should this be needed in order do guarantee the consistency of all layers. The newly introduced layer provides the developer with further information on the system. This information is used to improve the system being developed, to validate its prescriptive model, and to verify its properties. For example, the development of the system in simulation provides the developer with new data that can be used to improve and validate the prescriptive model and further verify that the desired properties hold.

Phase one: Properties – In this phase, the developer formally specifies the requirements of the robot swarm in the form of desired properties. These properties are the distinguishing features of the robot swarm that the developer wants to realize. They can be task specific, such as *the system eventually completes task X*, or they can express more generic properties, such as *the system keeps working as long as there are at least N robots* or *the system will never be in state Y for more than t time-steps*. The clearer and more complete these properties are in this phase, the more the developed robot swarm will meet expectations. Clearly stated requirements help reducing the risk of developing "the wrong robot swarm." For simplicity, we assume that requirements do not change during the development of the robot swarm.

Phase two: Model – In this phase, the developer creates a prescriptive model of the robot swarm. Usually, the prescriptive model describes how robots change state over time, where a state is an abstract simplified description of the actions of a robot (see also Section 2.2.1). The prescriptive model should be sufficiently detailed to capture the behavior of the robots and their interaction, but should not be too detailed, in order to avoid unnecessary complication.

Once a first draft of the prescriptive model is produced, the desired properties stated in phase one are verified using model checking. As in test-driven development (Beck, 2003), at first it is possible that the prescriptive model does not satisfy all the desired properties. In an iterative process, the developer expands and improves the prescriptive model, until the properties are satisfied. The outcome of this process is a prescriptive model of the collective behavior of the robot swarm that satisfies the stated properties.

Phase three: Simulation – In this phase, the developer uses the prescriptive model as a blueprint to implement and improve the robot swarm using a physics-based computer simulation (henceforth simply simulation). By blueprint we mean that the prescriptive model is used to identify the most relevant aspects of the robot swarm to realize. This allows the developer to focus on these aspects and neglect other minor details. For example, if a prescriptive model shows that, by entering state i, an individual robot affects the performance of the whole swarm more than by entering state j, the developer can focus on the first and temporarily ignore the second. Moreover, concentrating on the prescriptive model at design time allows the developer to direct his efforts towards high-level decisions rather than on the implementation.

It is possible that the implementation choices or other unforeseen aspects of the system yield in a simulated system that does not behaves as predicted by the prescriptive model. In this case the developer must go back to the previous phases, modify the prescriptive model to consider the results obtained from the simulation, and verify whether the required properties still hold true.

Phase four: Robots – In the last phase, the developer realizes the final robot swarm. Similarly to the transition between the prescriptive model and the simulation, if the implementation on robots reveals that some assumptions made during the previous phases do not hold, it might be necessary to modify the simulated version or the prescriptive model, in order to keep all levels consistent.

3.3 Case studies

In this section, we illustrate property-driven design using two very common case studies from the swarm robotics literature (Brambilla et al., 2013): aggregation and foraging.

In both case studies, we perform model checking using PRISM, a state-of-the-art suite for model checking (Kwiatkowska et al., 2004). PRISM is free and is released as open source software under the GNU General Public License (GPL).[1]

3.3.1 Aggregation

In the first case study, we tackle *aggregation*: robots have to cluster in an area of the environment. The robots have neither knowledge of the position of the other

[1]http://www.prismmodelchecker.org

robots nor a map of the environment. We choose aggregation as a case study for various reasons: i) aggregation is a simple case study and this allows us to focus on the development process; ii) aggregation is a common case study in swarm robotics (Brambilla et al., 2013); iii) aggregation possesses many of the salient traits of swarm robotics; it is completely distributed, it is based on simple robot-to-robot interactions, and it is characterized by stochasticity and spatial aspects.

The aggregation case study that we discuss in this section is similar to the one presented by Jeanson et al. (2005). We consider a dodecagonal environment with two black spots of equal size called *area A* and *area B*. We call *area C* the remaining white area. Each of the black spots is large enough to host all the robots. See Figure 3.5 for a picture of the environment. We consider three swarm sizes: 10, 20 and 50. We use three different arenas for the three different group sizes, respectively of 4.91 m^2, 19.63 m^2 and 50.26 m^2.

In the following, we will apply the 4-phase process explained in Section 3.2.

Phase one: Properties – The main property that the robot swarm must satisfy is *"eventually all the robots form an aggregate either on area A or area B"*. We set a time limit of 1000 seconds. Using PRISM syntax, we can define the following property:

$$P \geq k[F \leq 1000\ (S_a = N_t)|(S_b = N_t)] \tag{3.1}$$

In less formal terms, we want to know whether, in the first thousand seconds ($F \leq 1000$), the number of robots in area A (S_a) or in area B (S_b) is equal to the total number of robots in the swarm ($(S_a = N_t)|(S_b = N_t)$), with a probability greater or equal to k ($P \geq k$). The value of k depends on the size of the swarm: $k = 0.80$ for $N_t = 10$; $k = 0.40$ for $N_t = 20$; and $k = 0.01$ for $N_t = 50$;

Another property is that the aggregate, once formed, is stable for at least 10 seconds, that is, robots do not change state once the aggregate is formed. We want this to happen more than two thirds of the time an aggregate is formed:

$$(S_a = N_t)|(S_b = N_t) \Rightarrow P \geq 0.67\ [G \geq 10\ (S_a = N_t)|(S_b = N_t)] \tag{3.2}$$

In natural language, Property 3.2 can be expressed in this way: from the aggregate state ($(S_a = N_t)|(S_b = N_t)$) is it true with probability of at least 0.67 ($P \geq 0.67$) that the robot swarm stays for at least 10 seconds ($G \geq 10$) in the aggregate state?

Phase two: Model – To develop the prescriptive model for the aggregation

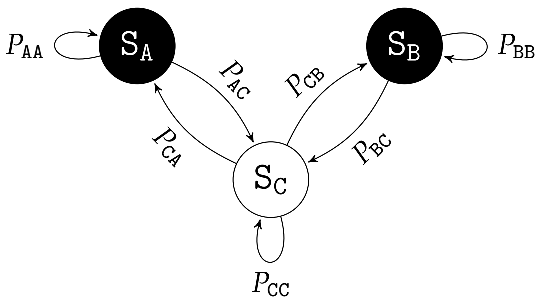


Figure 3.2: The prescriptive model for aggregation. Each state is used to count the number of robots in the corresponding area.

case study, we consider the three areas in which the environment is divided. We define three states: $\mathtt{S_a}$, $\mathtt{S_b}$ and $\mathtt{S_c}$. A robot in area A or B is in state $\mathtt{S_a}$ or $\mathtt{S_b}$, respectively. Robots outside area A or B are in state $\mathtt{S_c}$. We develop a discrete-time macroscopic prescriptive model. In a macroscopic model each state is associated with a counter to track the number of robots currently in that state (see Section 2.2.1). In this model, we have three counters, `a`, `b` and `c`, where `a+b+c=N_t`, associated to the respective states. See Figure 3.2 for the model of the system.

In this initial stage of the definition of the prescriptive model, we assume that the system can be effectively described by a *non-spatial* model, that is, a model in which the trajectories of the robots are ignored and a robot can move instantaneously from area C to area A or B, and vice versa. Moreover, for the moment, we also ignore the effects of interferences between robots (Lerman et al., 2005). In case these assumptions prove to be not realistic and the results obtained with the prescriptive model do not match those obtained in simulation or with the final robot swarm, we will modify them in the following phases, as explained in Section 3.2.

The first design attempt is the following: a robot performs random walk and, when it finds a black area, it stops. A robot stopped on a black area has a fixed probability to leave.

Since the prescriptive model is non-spatial and ignores interference, we consider only the geometric properties of the areas to compute $p_{\mathtt{CA}}$, that is, the instantaneous probability that a robot transitions from $\mathtt{S_C}$ to $\mathtt{S_A}$. A robot in area C can either go to area A, go to area B or stay in area C. This means that a robot in area C has a probability of going from area C to area A equal to $p_{\mathtt{CA}} = \frac{A_A}{A_{arena}}$, of going from area C to area B equal to $p_{\mathtt{CB}} = \frac{A_B}{A_{arena}}$, and of staying in area C

Table 3.1: Model checking results for the first solution with a fixed $p_{\texttt{AC}}$. Column $p_{\texttt{AC}}$ shows the best value of $p_{\texttt{AC}}$. Columns Pr 1 and Pr 2 show whether Property 3.1 and 3.2 are satisfied and the exact values of the probabilities involved in their definition

N_t	A_A	A_{arena}	$p_{\texttt{CA}}$	$p_{\texttt{AC}}$	Pr 3.1	Pr 3.2
10	$0.38\,\text{m}^2$	$4.91\,\text{m}^2$	0.08	0.05	X (0.75)	X (0.46)
20	$0.78\,\text{m}^2$	$19.63\,\text{m}^2$	0.06	0.04	X (0.15)	X (0.07)
50	$3.14\,\text{m}^2$	$50.26\,\text{m}^2$	0.06	0.04	X (8.8×10^{-5})	X (3.7×10^{-5})

equal to $p_{\texttt{CC}} = \frac{A_C}{A_{arena}} = 1 - (p_{\texttt{CA}} + p_{\texttt{CB}})$. Note that $p_{\texttt{CA}} = p_{\texttt{CB}}$, since the two areas have the same size.

The remaining probabilities depend on the behavior of the robots. The aggregate can be obtained in area A or area B, thus we set the probabilities of leaving these two areas to be equal: $p_{\texttt{AC}} = p_{\texttt{BC}}$. A robot in area A can only go to area C or stay in area A, thus $p_{\texttt{AA}} = 1 - p_{\texttt{AC}}$. The same holds for area B. From the above, it follows that $p_{\texttt{AA}} = p_{\texttt{BB}}$. The only independent probability remaining is $p_{\texttt{AC}}$. Through model checking, we can find the value of $p_{\texttt{AC}}$ that maximizes the probability involved in the definition of Property 3.1.

Using model checking, we can find the best values for parameter $p_{\texttt{AC}}$ and whether the required properties are satisfied. Using PRISM we can also compute the exact probabilities involved in the definition of the properties. In other words, we can use PRISM to answer the question: *"what is the probability that an aggregate is formed in less than 1000 seconds?"*. Table 3.1 shows that this first attempt at tackling the aggregation case study is unsuccessful. The behavior obtains poor results and the system does not cope well with increasing group sizes.

An analysis of the prescriptive model can help us in improving the developed behavior. From the obtained results we observed that a fixed $p_{\texttt{AC}}$ does not promote the formation of a single aggregate. A better solution is to let a robot decide whether to leave according to the number of sensed robots around it: with only few robots nearby, the probability to leave the aggregate $p_{\texttt{AC}}$ is high and vice versa. We set $p_{\texttt{AC}} = 1 - p_{min-\texttt{AC}} * (N_s + 1)$, where $p_{min-\texttt{AC}}$ is the minimum staying probability we want for a robot and N_s is the number of other robots sensed. We add 1 to the number of robots sensed, as we include also the robot that is choosing its next action. Subsequently, using model checking, we find the best value of $p_{min-\texttt{AC}}$ for the different group sizes. As reported in Table 3.2, results are significantly better both for Property 3.1 and Property 3.2.

With the current prescriptive model we are also able to define specifications of

Table 3.2: Model checking results for the second solution where $p_{\texttt{AC}} = 1 - p_{min-\texttt{AC}} * (N_s + 1)$. Column $p_{min-\texttt{AC}}$ shows the best value of $p_{min-\texttt{AC}}$. Column Pr 1 and Pr 2 are defined as in Table 3.1.

N_t	A_A	A_{arena}	$p_{\texttt{CA}}$	$p_{min-\texttt{AC}}$	Pr 3.1	Pr 3.2
10	$0.38\,\text{m}^2$	$4.91\,\text{m}^2$	0.08	$[0.19, 0.24]$	✓ (0.95)	✓ (0.92)
20	$0.78\,\text{m}^2$	$19.63\,\text{m}^2$	0.06	0.12	✓ (0.79)	✓ (0.87)
50	$3.14\,\text{m}^2$	$50.26\,\text{m}^2$	0.06	0.10	✓ (0.25)	✓ (0.71)

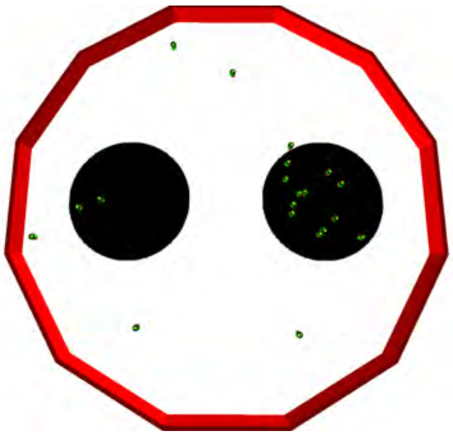


Figure 3.3: A screenshot of the simulated version of the robot swarm with 20 robots.

the hardware capabilities of the robots: a ground sensor, to differentiate between the two black areas A and B and the white area C; a sensor to detect nearby robots; and wheels to move. An example of such a robot is the e-puck (Mondada et al., 2009), which can be extended with a range and bearing board that allows it to perceive the presence of neighboring robots (Gutiérrez et al., 2009).

Phase three: Simulation – In this aggregation case study, the prescriptive model captures well the microscopic behavior of the single robots, thus it is quite straightforward to implement the robot swarm in simulation. However, several implementation details are not explicitly present in the prescriptive model, such as how the robots perform random walk, and have now to be programmed explicitly.

We implement the robot swarm using the ARGoS simulator (Pinciroli et al., 2012). Figure 3.3 presents a screenshot of the simulated robot swarm.

We perform three different sets of experiments, one for each group size. To validate the prescriptive model we measure the average time necessary to

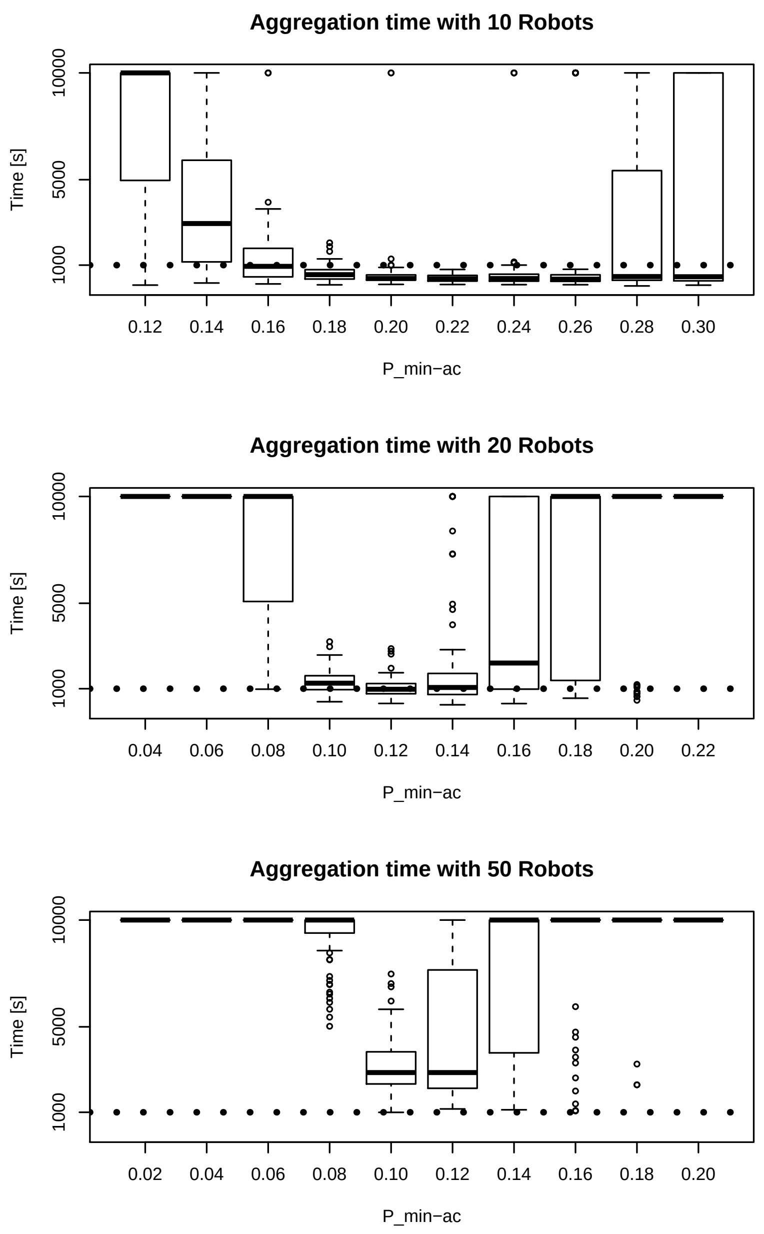


Figure 3.4: The results obtained with the ARGoS simulator. The graphs show the time at which the experiment is stopped. This time is less then 10,000 seconds in case the aggregate is formed, or equal to 10,000 seconds in case the aggregate is not formed. Results are presented for different $p_{min-\texttt{AC}}$ over 100 runs for 10, 20 and 50 robots.

form a complete aggregate on 100 runs with different values of $p_{min-\texttt{AC}}$. The robots are deployed in a random position at the beginning of each experiment. Each experiment is halted when a complete aggregate is formed or after 10,000 seconds.

As reported in Figure 3.4, for all the three group sizes, the best results are obtained with the value $p_{min-\mathtt{AC}}$ predicted using the prescriptive model. However, the results related to Property 3.1 obtained with the simulated version of the robot swarm are usually worse than those predicted by the prescriptive model, in particular with 20 and 50 robots. With 10 robots and $p_{min-\mathtt{AC}} = 0.22$ the simulated robot swarm was able to form a complete aggregate before 10,000 seconds 100 times out of 100, in line with the predictions of the prescriptive model. However, with 20 robots and $p_{min-\mathtt{AC}} = 0.12$, an aggregate was formed in less than 1,000 seconds only 53 times out of 100, whereas in the prescriptive model this happened with probability 0.79. With 50 robots and $p_{min-\mathtt{AC}} = 0.10$ the difference is even more evident: only 2 runs out of 100 resulted in an aggregation time of under 1,000 seconds whereas the prescriptive model predicted a probability of 0.25.

As explained in Section 3.2, since the results obtained from the prescriptive model do not match those obtained with simulations, we need to modify the model in order to make them consistent. Our conjecture is that the discrepancy in performance between the prescriptive model and the simulated robot swarm is due to the fact that, as the number of robots grows, interference between robots reduces $p_{\mathtt{CA}}$. This is because the robots spend time avoiding collisions and because the robots stopping in the black areas prevent other robots from accessing them. These aspects are not considered explicitly in the model. Reducing $p_{\mathtt{CA}}$ in the model allows us to obtain results that are closer to those obtained in simulation. For 10 robots there is no need to modify $p_{\mathtt{CA}}$, as the results already match. For 20 robots and $p_{\mathtt{CA}} = 0.05$, we observe that Property 3.1 is satisfied: robots form an aggregate in less than 1,000 seconds with probability 0.53. This matches the results obtained in simulation. For 50 robots we set $p_{\mathtt{CA}} = 0.04$, which gives a probability of 0.01.[2] Table 3.3 presents a comparison between the number of successful aggregates obtained before 1,000 seconds obtained in simulation and those obtained with model checking with the old and $p_{\mathtt{CA}}$ new values.

To test Property 3.2, we perform 100 runs of the simulated experiments for 10,000 seconds with the three group sizes. In the experiments, we measure whether the robot swarm satisfies Property 3.2, that is, whether a complete

[2] To obtain better results with 50 robots it would be necessary to develop one behavior for $N_t < 50$ and a different behavior for $N_t \geq 50$. Since our main goal is to introduce property-driven design, for the sake of simplicity we avoid this. Note however that, as showed in Figure 3.4, using our approach we were able to find the best parameter for $N_t = 50$.

Table 3.3: A comparison between model checking and simulation. The table presents the probability involved in the definition of Property 3.1 (model checking) compared to the experimental results over 100 runs (simulation).

N_t	Model checking (old $p_{\mathtt{CA}}$ val.)	Model checking (new $p_{\mathtt{CA}}$ val.)	Simulation
10	0.95 with $p_{\mathtt{CA}} = 0.08$	0.95 with $p_{\mathtt{CA}} = 0.08$	100/100
20	0.79 with $p_{\mathtt{CA}} = 0.06$	0.53 with $p_{\mathtt{CA}} = 0.05$	53/100
50	0.25 with $p_{\mathtt{CA}} = 0.06$	0.01 with $p_{\mathtt{CA}} = 0.04$	2/100

aggregate, once formed, lasts more than 10 seconds. In all the cases in which a complete aggregate was formed before 10,000 seconds, Property 3.2 was satisfied.

Videos of the simulated experiments are available in the supplementary material (Brambilla et al., 2014b).

Phase four: Robots – We perform 10 experiments with a group of 10 e-pucks in an arena identical to the simulated one. A screenshot of an experiment can be seen in Figure 3.5. Figure 3.6 shows a comparison between the time necessary for achieving aggregation obtained with the robots and in simulation. A video of a run is available in the supplementary material (Brambilla et al., 2014b).

In 10 runs out of 10, both Property 3.1 and Property 3.2 were satisfied. The results obtained with the robots are in line with those obtained in simulation.

The obtained robot swarms is able to aggregate satisfying the required properties. For this reason, there is no need to further update the prescriptive model and we can declare the process completed.[3]

3.3.2 Foraging

In the second case study, we tackle foraging. In the simplest form of foraging, robots harvest *objects* and store them in the *nest*. The objects can be scattered in random positions or located in specific areas in the environment called *sources*. Foraging can be seen as an abstraction of more complex and realistic applications, such as search and rescue, land mine removal, waste cleaning and automated warehouse operation.

The number of objects retrieved typically depends on the number of robots: a single robot can perform foraging alone, but additional robots could be added to increase the performance of the swarm as robots working in parallel are able to retrieve more objects per time unit than a single robots. However, when the

[3]The obtained results are satisfying for $N_t = 10$ and $N_t = 20$

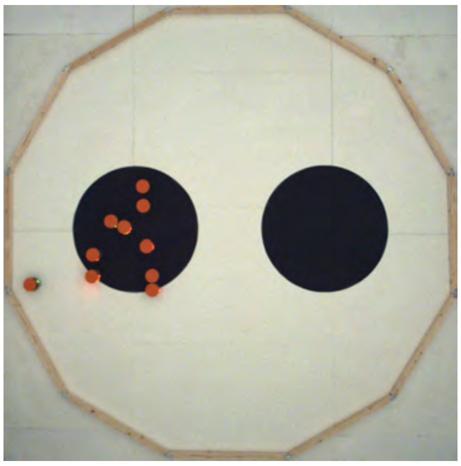


Figure 3.5: A picture of an experiment performed with 10 e-puck robots.

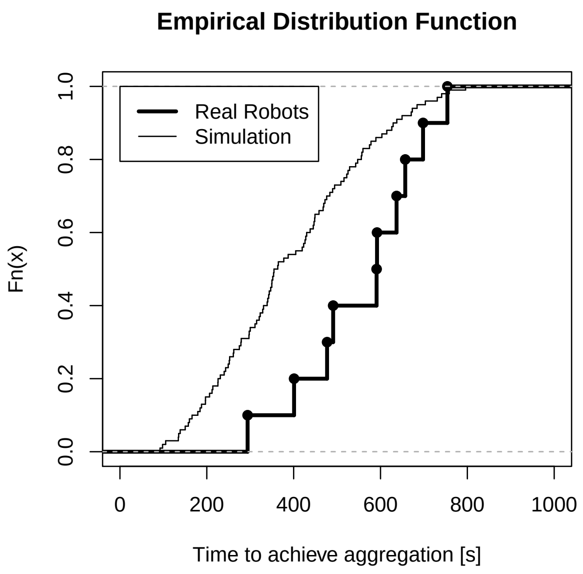


Figure 3.6: A graph showing the empirical cumulative distribution $Fn(x)$ of the time necessary to achieve aggregation obtained with robots (10 runs) and in simulation (100 runs). In both cases $N_t = 10$ and $p_{min-\texttt{AC}} = 0.22$.

density of the robots in the environment increases, the performance of each single robot may be reduced due to interference (Lerman and Galstyan, 2002, Pini et al., 2009).

In this case study, we assume that the robotic platform is given: the task must be tackled using e-pucks (Mondada et al., 2009). The e-puck does not have the manipulation capabilities to interact with physical objects, so we consider an abstract version of foraging: instead of interacting with objects, e-pucks interact with TAM devices (Brutschy et al., 2010). The TAM is a device similar to a booth, in which a robot can enter. It has a system of light barriers to sense the presence of a robot, and an LED that can be used to communicate information about its internal state. In this case study, TAMs are used to simulate the manipulation of objects: an e-puck can enter in a TAM, wait a fixed time and leave to simulate harvesting or storing an object.

The arena comprises 20 TAMs: 5 TAMs on the north wall act as the nest, each of these TAMs is a storing location; 15 TAMs on the other walls act as sources, locations where objects can appear. At any given time, in the arena there are O objects available, that is, a new object appears as soon as one is harvested by a robot. We perform experiments in which O equals $\{2, 4, 6, 8, 10\}$. The number of available storing locations depends on the number of robots currently storing an object: it can vary from 5, when no robot is using a storing location, to 0 if all are in use.

The state of a TAM is encoded using colors: green when the TAM is available for storage; blue when the TAM has an object available for harvesting; red when the TAM is busy, that is, a TAM in which a robot is currently harvesting or storing an object; off/black when the TAM is unavailable.

The environment is enclosed in $2\,\mathrm{m} \times 2\,\mathrm{m}$ square (see Figure 3.7 and Figure 3.12). Note that there is no globally perceivable clue in the environment that informs the robots of the position of the nest, differently from many other foraging studies (See Brambilla et al. (2013) for a review including work on foraging).

To allow robots to see the TAMs, we use e-pucks equipped with an omnidirectional camera.[4] Using the omnidirectional camera, robots can see the LEDs of the TAMs within a range of $0.5\,\mathrm{m}$.

In the following, we will apply the 4-phase process presented in Section 3.2. In the foraging case study, we use the continuous time version of the Markov chain model, to model more easily the duration of some actions, such as harvesting

[4]See http://www.gctronic.com for more details.

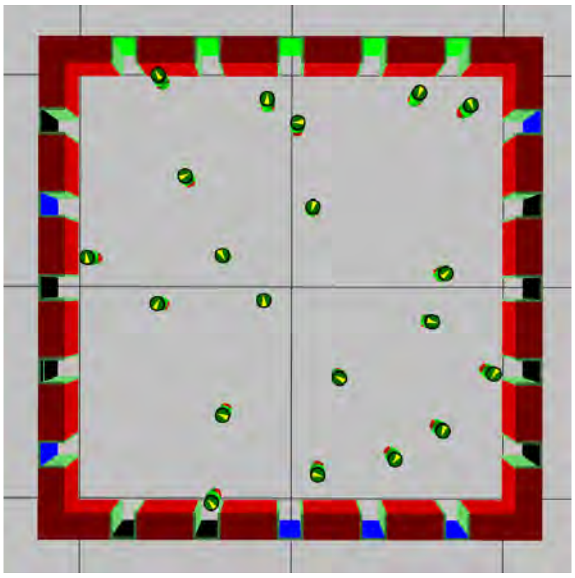


Figure 3.7: A screenshot of the simulated version of foraging using 20 robots. Green lighted TAMs signal storage locations, blue lighted TAMs signal objects to be taken, dark TAMs are not available.

and storing an object.

Phase one: Properties – In foraging, the main requirement is that the swarm retrieves at least a certain number of objects within a fixed time:

$$R_{obj\_ret} \geq k \ [C \leq 600] \tag{3.3}$$

where R_{obj_ret} indicates that we are interested that the expected value of the reward *obj_ret* is greater or equal than k, and $C \leq 600$ indicates that we are interested in the cumulative value over 600 seconds. The number k of objects that we wish to retrieve depends on N_t, the number of robots composing the swarm, and on O, the number of objects available in the environment at any given time; see Table 3.4.

Another requirement is on the *worst case performance*, that is, we want to ensure that the robot swarm is able to retrieve at least a minimum number of objects in 600 seconds. Model checking allows us to formally verify this condition since we can compute not only the expected value, but also its cumulative distribution (or, conversely, the density function). Formally, the second requirement is defined as:

$$P \geq 0.90 \ [F \leq 600 \, obj\_ret > 40] \tag{3.4}$$

Table 3.4: The value of k necessary to satisfy Property 3.3 for different values of N_t, the number of robots composing the swarm, and values of O, the number of objects available in the environment at any given time.

N_t	O	k
20	2	45
20	4	55
20	6	65
20	8	75
20	10	85

N_t	O	k
10	6	40
20	6	65
50	6	90
100	6	75

In natural language, Property 3.4 can be expressed as: is it true with probability greater than 0.90 ($P \geq 0.90$) that at least 40 objects are retrieved ($obj\_ret > 40$) in less than 600 seconds ($F \leq 600$)? To simplify the discussion, we verify Property 3.4 only in the case where $N_t = 20$ and $O = 6$.

Phase two: Model – To build the prescriptive model, we consider the different actions that a robot must perform. We then associate a state of the Markov chain to each of these actions.

A robot searches for objects by performing random walk in the environment (`So` state). Once an object is found, the robot tries to harvest it (`H` state); in case of multiple objects in range, the robot goes towards the closest one. If the harvest action is unsuccessful, because, for instance, another robot harvests the object, the robot goes back to searching. When the object is reached, the robot waits inside the TAM for a fixed amount of time until the object is harvested (`Hw` state). Once the robot has harvested an object, it proceeds to search for the nest by performing random walk (`Sn` state). As soon as an available storage location is found, the robot tries to store the carried object (`ST` state); also in this case, the closest storage location is approached if multiple storing locations are seen. If the store action is unsuccessful, the robot searches for another storage location until the object is stored. Similar to the harvest operation, also in this case the robot waits inside a TAM for a fixed amount of time until the object is stored (`STw` state). A successful store operation increases the object counter (`obj_ret`). The robot then searches for a new object to harvest.

Robots always try to avoid collisions with obstacles and other robots. Practically, this produces two behaviors: when a robot is trying to enter a TAM (state `H` or state `ST`), it follows a vector that is the sum of a vector pointing to the desired

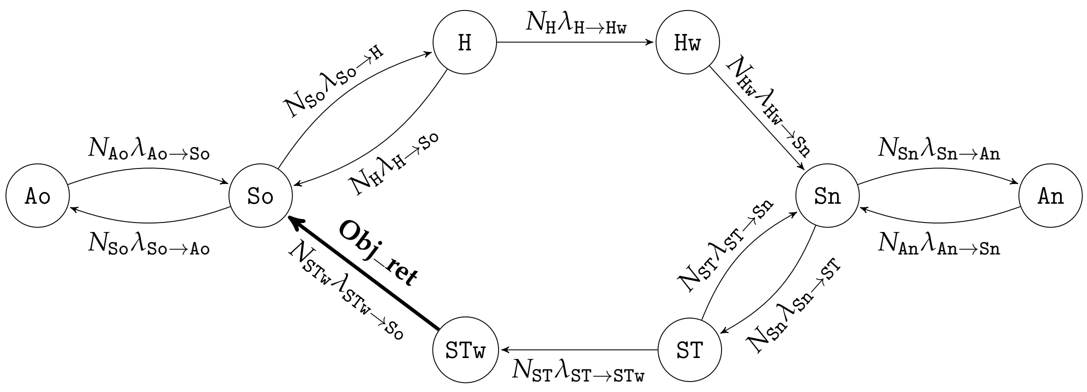

H
Hw
Ao
So
Sn
An
STw
ST
$N_{\mathtt{H}}\lambda_{\mathtt{H}\to\mathtt{Hw}}$
$N_{\mathtt{So}}\lambda_{\mathtt{So}\to\mathtt{H}}$
$N_{\mathtt{H}}\lambda_{\mathtt{H}\to\mathtt{So}}$
$N_{\mathtt{Hw}}\lambda_{\mathtt{Hw}\to\mathtt{Sn}}$
$N_{\mathtt{Ao}}\lambda_{\mathtt{Ao}\to\mathtt{So}}$
$N_{\mathtt{So}}\lambda_{\mathtt{So}\to\mathtt{Ao}}$
$N_{\mathtt{Sn}}\lambda_{\mathtt{Sn}\to\mathtt{An}}$
$N_{\mathtt{An}}\lambda_{\mathtt{An}\to\mathtt{Sn}}$
Obj_ret
$N_{\mathtt{STw}}\lambda_{\mathtt{STw}\to\mathtt{So}}$
$N_{\mathtt{ST}}\lambda_{\mathtt{ST}\to\mathtt{Sn}}$
$N_{\mathtt{Sn}}\lambda_{\mathtt{Sn}\to\mathtt{ST}}$
$N_{\mathtt{ST}}\lambda_{\mathtt{ST}\to\mathtt{STw}}$

Figure 3.8: The continuous time Markov chain used to model foraging. H is the *harvest* state; Hw is the *wait to harvest* state; *So* is the *search object* state; Ao is the *avoid (while searching for an object)* state; ST is the *store* state; STw is the *wait to store* state; Sn is the *Search nest* state; An is the *Avoid (while searching for the nest)* state; Transitions are labeled with their respective rates multiplied by the number of robots currently in that state: $\lambda_{\mathtt{i}\to\mathtt{j}}$ is the rate at which an individual robot moves from state i to state j; $N_{\mathtt{i}}$ is the number of robots currently in state i. To compute the expected number of objects retrieved, we keep track of the number of times the transition from STw to So, labeled **Obj_ret**, happens.

destination and a vector pointing away from the closest obstacle. When a robot is performing random walk without a specific destination instead (state So or state Sn), if it encounters an obstacle or another robot, it starts turning on the spot for a random number of steps and then it begins again to move straight. This random number of steps follows a geometrical distribution. We model these different reactions in two different ways: in the first case, the action of the robot is not significantly disturbed, as the robot performs only a slight change of trajectory towards its goal. For this reason, this first kind of collision avoidance affects only the time to complete the action, but does not change the behavior of the robot. In the second case, instead, the robot completely changes its direction to avoid a collision, resulting in a significant change in its behavior and its chance to find objects or the nest. For this reason, the second kind of collision avoidance is modeled by adding two states: state Ao in case the robot is avoiding a collision when searching for an object, and state An in case the robot is avoiding a collision when searching for the nest.

See Figure 3.8 for a complete view of the prescriptive model.

We now have the structure of the behavior that the robots should follow. We need to assign values to the transition rates. We compute the transition rates considering the behavior of a single robot. For the macroscopic model, the rates are then multiplied by the current number of robots in the related state, as illustrated in Figure 3.8.

All the rates involved in the definition of the model depend on the geometrical characteristics of the environment and/or on the behavior of the robots. Unfortunately, differently from the previous case study, we cannot completely define them a priori since it is impossible to identify the correct value of the parameters involved in their definition without experimental data. Note that the goal of this phase is not to create a model that is as precise as possible, but one that can be used by us to develop and improve the desired robot swarm. Since we do not have experimental data, the model we are creating is largely arbitrary. Other valid choices could have been made. We make some working hypotheses about the system that can be subject to refinements or changes in the subsequent phases, should they prove not to be sufficiently accurate or correct. In particular, parameters will be fitted once experimental data are available, that is, in phase three.

In the following, we present how each rate is defined. We define $\lambda_{\texttt{So}\rightarrow\texttt{H}}$, the rate at which a robot finds an object, as proportional to the density of available objects in the environment:

$$\lambda_{\texttt{So}\rightarrow\texttt{H}} = \alpha \frac{O}{A_{arena}},$$

where O is the number of objects available at any given time, A_{arena} is the area of the environment and α is a parameter.

We define $\lambda_{\texttt{So}\rightarrow\texttt{Ao}}$, the rate at which a robot searching for an object finds another robot and then performs obstacle avoidance, as proportional to the density of robots in the environment:

$$\lambda_{\texttt{So}\rightarrow\texttt{Ao}} = \beta \frac{N_t}{A_{arena}},$$

where N_t is the total number of robots in the environment, and β is a parameter.

Rates $\lambda_{\texttt{Sn}\rightarrow\texttt{ST}}$ and $\lambda_{\texttt{Sn}\rightarrow\texttt{An}}$ are defined similarly, considering the number of storage locations instead of the number of objects:

$$\lambda_{\texttt{Sn}\rightarrow\texttt{ST}} = \gamma \frac{D}{A_{arena}},$$

$$\lambda_{\texttt{Sn}\rightarrow\texttt{An}} = \delta\frac{N_t}{A_{arena}},$$

where D is the number of storage locations and γ and δ are two parameters.

Rates $\lambda_{\texttt{Ao}\rightarrow\texttt{So}}$ and $\lambda_{\texttt{An}\rightarrow\texttt{Sn}}$ depend on the time necessary for a robot to perform obstacle avoidance. As said before, if a robot encounters an obstacle or another robot while searching for objects or for the nest, it performs collision avoidance by turning on the spot for a random number of steps distributed geometrically and then it starts again searching for objects or the nest. The rate at which a robot moves from collision avoidance back to search is thus:

$$\lambda_{\texttt{Ao}\rightarrow\texttt{So}} = \lambda_{\texttt{An}\rightarrow\texttt{Sn}} = p_{oa},$$

where p_{oa} is the parameter of the geometrical distribution.

We define $\lambda_{\texttt{H}\rightarrow\texttt{Hw}}$ and $\lambda_{\texttt{ST}\rightarrow\texttt{STw}}$, the rates at which a robot going towards an object-TAM or a storage-TAM manages to enter it, as the reciprocal of the time necessary to get in the TAM, counted from the instant in which the robot sees it:

$$\lambda_{\texttt{H}\rightarrow\texttt{Hw}} = \lambda_{\texttt{ST}\rightarrow\texttt{STw}} = \left(\frac{r}{s}\right)^{-1},$$

where r is the range at which a robot sees a TAM and s is the forward speed of a robot.

A robot trying to enter a TAM is not always successful, other robots may "steal" its object by occupying the storage location before it can do it. This means that not all robots going towards a TAM enter it. Some are interrupted by other robots and thus are forced to search for another available TAM. This is modeled by the transition $\texttt{H} \rightarrow \texttt{So}$ and $\texttt{ST} \rightarrow \texttt{Sn}$. We define $\lambda_{\texttt{H}\rightarrow\texttt{So}}$ and $\lambda_{\texttt{ST}\rightarrow\texttt{Sn}}$, the related rates, as proportional to the density of robots in the environment:

$$\lambda_{\texttt{H}\rightarrow\texttt{So}} = \epsilon\frac{N_t}{A_{arena}},$$

$$\lambda_{\texttt{ST}\rightarrow\texttt{Sn}} = \eta\frac{N_t}{A_{arena}},$$

where ϵ and η are parameters. We expect ϵ and η to have different values, as storage locations are all next to each other, generating more interference, while objects to harvest in general are evenly distributed along the walls of the environment.

The last rates we need to define are $\lambda_{\texttt{Hw}\rightarrow\texttt{Sn}}$ and $\lambda_{\texttt{STw}\rightarrow\texttt{So}}$, the rates at which robots in the TAM complete their operations and exit. These rates are the reciprocal of the time spent by a robot in a TAM:

$$\lambda_{\texttt{Hw}\to\texttt{Sn}} = \lambda_{\texttt{STw}\to\texttt{So}} = (t_{TAM})^{-1},$$

where t_{TAM} is the time spent by a robot in a TAM.

Since we do not have empirical data to estimate the parameters, at this point we cannot use model checking to compute the expected number of objects retrieved, or whether the desired properties are satisfied. However, even without empirical data, we can use the model to improve the behavior of the robots. For example, by analyzing the model, we can observe that increasing the rate at which the robots find objects or storage locations—that is, increasing $\lambda_{\texttt{So}\to\texttt{H}}$ and $\lambda_{\texttt{Sn}\to\texttt{ST}}$—results in an increase of objects retrieved.

In order to increase these rates, we cannot modify the number of objects available at any given time or the number of storage locations, since they are given. We could act on the parameters, but it is not clear how to change the behavior of the robots to increase these parameters. Even though we cannot change the dimensions of the environment, we can change the size of the area effectively covered by the robots. In other terms, we can change the behavior of the robots so that they do not cover the whole environment when searching for objects or storage locations. In particular, we could let robots avoid places where they know they will not find anything useful.

In the behavior defined before, robots searching for objects and for the nest go straight until they find an obstacle such as a wall or another robot. This means that robots that are carrying an object while searching for the nest may go close to other available objects, interfering with robots not carrying objects. Similarly, robots searching for objects often go close to the storing locations, interfering with the other robots. A possible solution is that robots searching for objects avoid storing locations as soon as they see them and, similarly, robots searching for the nest avoid objects as soon as they see them. This improved behavior is depicted in Figure 3.9.

This improvement in the behavior can be modeled by decreasing the value of A_{arena}. The rates are updated in the following way:

$$\lambda_{\texttt{So}\to\texttt{H}} = \alpha\frac{O}{A_o},$$

where A_o is the area explored by the robots searching for objects and avoiding

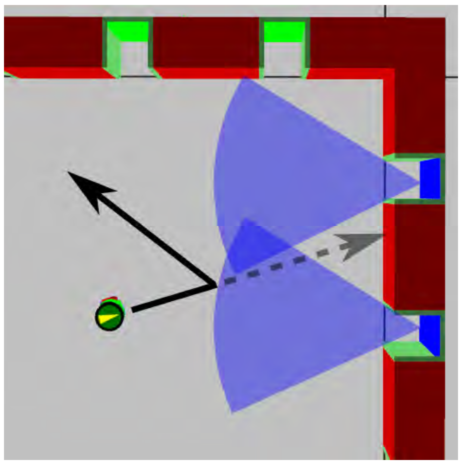


Figure 3.9: The modification of the behavior used to reduce interference and reduce the area searched by the robot. A robot (depicted as a green circle with a yellow arrow on top) carrying an object performs collision avoidance (full arrow) as soon as it sees an other object, instead of reaching to the wall (dotted arrow). The light blue circular sections represent the areas in which a robot sees an object. The same applies to robot searching for the nest, even though it is not displayed in the figure.

storage locations, with $A_o < A$;

$$\lambda_{\mathtt{Sn}\to\mathtt{ST}} = \gamma \frac{D}{A_n},$$

where A_n is the area explored by the robots searching for storage locations and avoiding objects, with $A_n < A$; All other rates are left unchanged since the density of the robots involved in their definition does not change. For example, consider $\lambda_{\mathtt{So}\to\mathtt{Ao}}$. The area involved in the definition of this rate is reduced, as explained above. However, also the number of robots operating in that area is reduced. In other words, even though the area considered is reduced, the density of robots does not change.

More complex improvements, such as task allocation mechanisms, could be implement to further increase the performance of the system, should the obtained performance not be sufficient. However, for the sake of brevity and clarity we limit our design process to the simple improvement presented above.

Phase three: Simulation – In this phase, we implement the foraging robot swarm using the ARGoS simulator (Pinciroli et al., 2012).

The prescriptive model developed in phase two provides us with a detailed blueprint to implement the robot swarm: the behavior of the individual robot

Table 3.5: The numerical parameters used in the foraging prescriptive model. The values are obtained from the experimental data obtained in phase three.

	Value		Value
O	$\{2, 4, 6, 8, 10\}$	p_{oa}	0.20
N_t	$\{10, 20, 50, 100\}$	r	$0.5\,\mathrm{m}$
D	5	s	$0.1\,\mathrm{m\,s^{-1}}$
A_{arena}	$4\,\mathrm{m}^2$	ϵ	6.5×10^{-2}
α	4×10^{-2}	η	1.01×10^{-1}
β	1×10^{-2}	$t_{\mathtt{TAM}}$	$3\,\mathrm{s}$
γ	3.9×10^{-2}	A_o	$3\,\mathrm{m}^2$
δ	9×10^{-3}	A_n	$3.7\,\mathrm{m}^2$

can be implemented using a finite state machine that resembles the Markov chain defined in phase two. Nonetheless, some implementation details, such as how robots stop inside a TAM, have been ignored in the prescriptive model, in order to focus on the more important details at design time and have to be programmed explicitly at this moment.

We measured the number of objects retrieved over 600 seconds on 100 runs. We first performed experiments using the initial behavior and then using the improved one, as explained in the previous phase. Table 3.5 shows the parameters used for model checking derived from the experimental data.

We can now compute the expected number of objects retrieved in 600 seconds using model checking on the developed model and compare these values with the results obtained from the simulated experiments.

Figure 3.10 shows the results obtained in simulation together with the expected results predicted by the prescriptive model. These results have been obtained using 20 robots with different values of O, the number of objects available at any time. Figure 3.11 shows the results obtained with $O = 6$ and a different number of robots. Table 3.6 shows whether Property 3.3 is satisfied.

In Figure 3.10 it is possible to observe that indeed the behavior improvement introduced in phase two significantly increases the number of objects retrieved. The improved behavior is always significantly better than its counterpart—Wilcoxon test with $p < 0.01$.

The correspondence between the results obtained from the prescriptive model and the ones obtained from the simulations is quite good, even though not perfect. For our goals and purposes, the model captures qualitatively the behavior of the robot swarm, thus it is not necessary to further refine it.

We also verify Property 3.4 using model checking and compare it with the

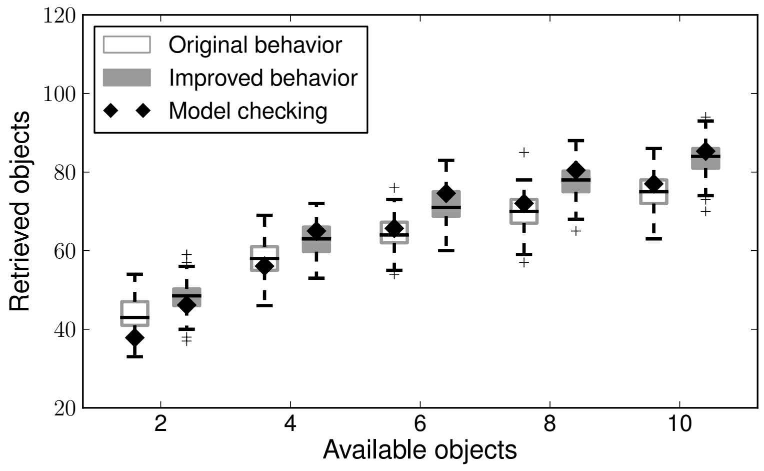


Figure 3.10: A comparison between the results obtained using 20 robots with the original behavior and with the improved one for different values of O, which is the number of objects available at any time. Box plots show results obtained over 100 experimental runs using the ARGoS simulator, while diamonds show the expected results obtained with PRISM.

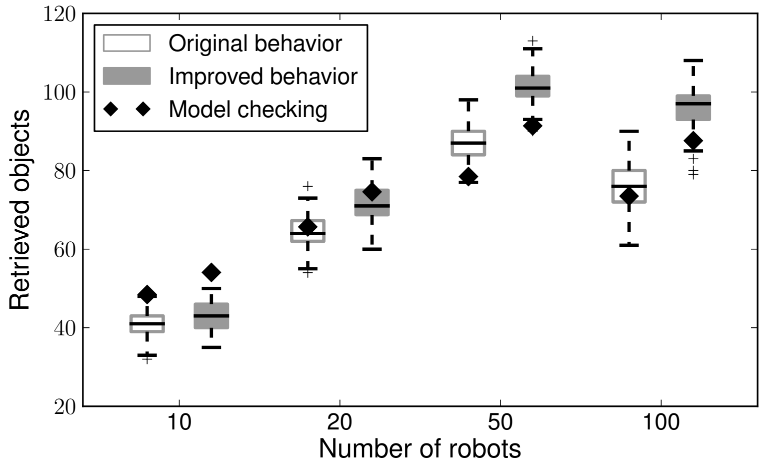


Figure 3.11: The number of object retrieved with different swarm sizes and $O = 6$. Box plots show results obtained over 100 experimental runs using the ARGoS simulator, while diamonds show the expected results obtained with PRISM.

Table 3.6: A table presenting whether the simulated systems are able to satisfy Property 3.3. k is the threshold on the expected number of objects retrieved. Property 3.3 is satisfied if the values obtained in simulations are greater or equal than k. The results are presented both for the original and the improved behavior, showing whether they satisfy Property 3.3 and the value of the median.

N_t	O	k	Original	Improved
20	2	45	X (43)	✓ (48)
20	4	55	✓ (58)	✓ (63)
20	6	65	X (64)	✓ (71)
20	8	75	X (70)	✓ (78)
20	10	85	X (75)	✓ (85)

N_t	O	k	Original	Improved
10	6	40	✓ (41)	✓ (43)
20	6	65	X (64)	✓ (71)
50	6	90	X (87)	✓ (101)
100	6	75	✓ (76)	✓ (97)

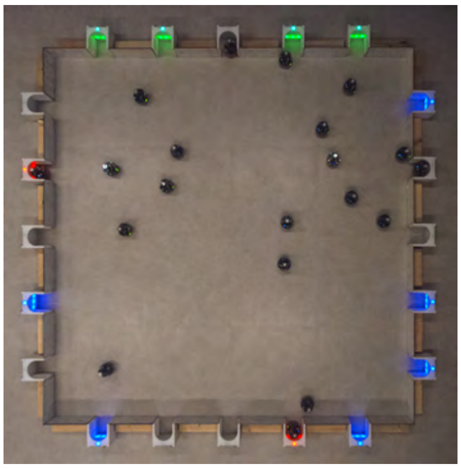


Figure 3.12: A picture of an experiment performed with 20 e-puck robots and $O = 6$.

results obtained from the simulations. Model checking tells us that Property 3.4 is not satisfied in the prescriptive model of the original behavior with 20 robots and $O = 2$. This matches the experimental results, where 15 runs over 100 resulted in less than 40 objects retrieved. Instead, with the improved behavior, property 3.4 is satisfied. This matches the experimental results, where only 2 runs over 100 resulted in less than 40 objects retrieved.

All experimental data can be found in the supplementary material (Brambilla et al., 2014b).

Phase four: Robots – We performed 10 experiments with a group of 20 e-pucks in an arena identical to the simulated one. A picture of an experiment can be seen in Figure 3.12. Videos of the performed experiments can be found

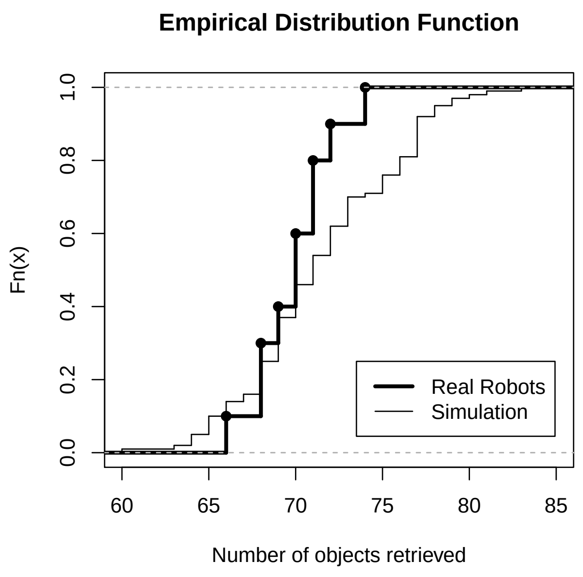


Figure 3.13: A graph showing the empirical cumulative distribution $Fn(x)$ of the number of object retrieved using robots (10 runs) and in simulation (100 runs). In both cases $N_t = 20$ $O = 6$.

in the supplementary material (Brambilla et al., 2014b). Figure 3.13 shows that the results obtained with real robots and simulated robots are very similar. Property 3.4 is satisfied. There is no need to update the model and thus we can declare the process completed.

3.3.3 Discussion

The two case studies presented in the section show that using property-driven design we were able to develop two robot swarms that tackle successfully aggregation and foraging: all the required properties are satisfied.

As shown, with property-driven design it is possible to analyze and develop behaviors characterized both by numerical and non-numerical parameters. For example, in the aggregation case we analyzed the effects of changing the probability to leave a black area, whereas in the foraging case we analyzed the effects of changing the exploration behavior.

One of the main advantages of property-driven design is that it allows the developer to focus on the design of the system rather than on its implementation: by focusing the designing efforts on the abstract model of the system, the developer can concentrate on the important aspects of the system to create because “whereas a simulation should include as much detail as possible, a good model should include as little as possible” (Smith, 1978). As an example, in the foraging case study we leveraged the prescriptive model developed in phase two of property-driven design to identify possible improvements of the partially realized system: we were able to identify the important aspects of the system being designed, which allowed us to avoid wasting time on improving other non-relevant aspects.

Another advantage of property-driven design is the reduced risk of developing a robot swarm that does not satisfy the requirements. Up to now, there was no clear way to specify the requirements of a robot swarm. In property-driven design, requirements are specified in a formal way at the beginning of the development process in terms of desired properties. Moreover, thanks to model checking it is possible to evaluate whether the robot swarm fulfills such properties at each step of the design and development process. This advantage of property-driven design has been highlighted by both case studies.

Finally, property-driven design addresses also the problem of the low reusability of solutions in swarm robotics. Usually behaviors for robot swarms are developed in a disposable way. This is due to the fact that there is no clear distinction between the design and the implementation. Thus, if a different hardware platform is available, or a slightly different task is tackled, it is necessary to start from scratch. With property-driven design instead, the prescriptive model developed in phase two can be partially or completely reused: i) the model is hardware independent, so that it can be adapted to the available robots, or even guide the process of deciding the best robot to use; and ii) the model can be extended to deal with new properties and verify if they are satisfied even without testing the system in simulation or with robots. The reusability of the prescriptive model reduces the risk that designers “reinvent the wheel” each time they develop a robot swarm. For example, the model of foraging developed in the presented case study could be easily adapted for robots with more sofisticated manipulation capabilities. In the future, it is also possible to imagine a set of publicly available models for swarm robotics applications that can be reused and modified by other developers.

The development process of the case studies presented in the section highlights also some issues with property-driven design.

The main issue is that ultimately the step from the prescriptive model to its implementation remains in the hands of the developer. Nonetheless, the prescriptive model can be used as a blueprint for the implementation process, providing the developer with a valuable tool to obtain robot swarms with provable properties.

Another issue is the strong reliance on modeling. Modeling robot swarms is a difficult task on its own: robot-to-robot interactions, spatial and temporal features and interference are difficult to completely describe using models. Luckily, modeling robot swarms has been the focus of a large number of studies (see two reviews of the literature by Brambilla et al. (2013) and Lerman et al. (2005)) providing a solid theoretical foundation to property-driven design.

3.4 Summary

Property-driven design is a top-down design method based on prescriptive modeling and model checking: the desired robot swarm is first described using a set of properties; subsequently a prescriptive model of the robot swarm is created; the prescriptive model is used as a blueprint for the implementation of the robot swarm first in simulation and then with robots.

Property-driven design is conceived to be part of swarm engineering: the systematic application of scientific and technical knowledge to specify requirements, design, realize, verify, validate, operate and maintain a swarm intelligence system. Up to now, the design and development of a robot swarm is performed using a code-and-fix approach based completely on the ingenuity and experience of the developer who does not have any scientific or technical support in his activity. Property-driven design aims at providing such scientific and technical support, with many advantages compared to the traditional unstructured approach.

In this section, we demonstrated, by tackling two different case studies, that property-driven design is an effective method for the design and development of robot swarms.

Chapter 4

Analysis of robot swarms using model checking

In this chapter, we demonstrate the use of model checking to analyze and verify robot swarms.

In Section 4.1, we analyze a collective decision-making behavior using a macroscopic Markov chain model developed with Bio-PEPA.

In Section 4.2, we analyze a collective transport behavior using a microscopic Markov chain model developed with KLAIM.

In Section 4.3, we provide a discussion concerning the use of model checking for the analysis of robot swarms.

This chapter is based on Massink et al. (2012, 2013), Gjondrekaj et al. (2012).

4.1 Analysis of a collective decision-making behavior using Bio-PEPA

As presented in Chapter 2, the most common ways to analyze a robot swarm is through the use of *stochastic simulations* (Dixon et al., 2011) and *fluid flow analysis* (Lerman et al., 2005). In this dissertation, we present the use of *model checking* as an effective analysis tool in swarm robotics.

Stochastic simulations, fluid flow analysis and model checking allow a developer to perform different analyses of a system behavior. However, for each of these analyses, a specific model is, in general, necessary: stochastic simulation is usually performed on microscopic models, fluid flow analysis is usually performed on macroscopic models based on rate equations, and model checking is usually performed on Markov chains. This means that, in order to perform a complete analysis of a robot swarm, it would be necessary to produce three different models. This model multiplication would greatly increase the effort necessary for the analysis process. Moreover, when dealing with different models, the issue of mutual consistency must be addressed, as different models are, in general, syntactically and semantically different. As a consequence, in the great majority of the cases only one model is produced.

In this section of this dissertation, we present a novel approach to model robot swarms based on Bio-PEPA (Ciocchetta and Hillston, 2009), which allows one to perform different consistent analyses of a system from the same formal specification.

Bio-PEPA is a process algebra originally developed for biochemical systems. It has been adopted to analyze a number of biological systems (Ciocchetta and Hillston, 2012) and disease spread (Benkirane et al., 2012), and it has also been used to analyze emergency egress (Massink et al., 2011a) and crowd dynamics (Massink et al., 2011b). These two last applications are particularly interesting for swarm robotics, as they concern systems characterized by a high number of individuals and lack of a centralized controller.

Bio-PEPA is well suited to analyze and develop robot swarms; with Bio-PEPA it is possible to develop a specification at the macroscopic level while providing also primitives for spatial and temporal description and for the composition of robot into teams or groups. Moreover, Bio-PEPA enables the developer to define *species*, which can be used to characterize groups of robots with specific attributes and actions; for instance, *species* can be used to differentiate between groups of robots performing different tasks at the same location.

In this dissertation, we use Bio-PEPA to further analyze a collective decision-making behavior that has been studied in a number of other papers (Montes de Oca et al. (2011), Scheidler (2011) and Valentini et al. (2012)). The case study consists of a robot swarm that have to identify the shortest path between two possible alternatives. We validate our results against those presented in Montes de Oca et al. (2011).

The outline of this section is as follows. In Section 4.1.1, we give a brief presentation of Bio-PEPA. In Section 4.1.2, we present the case study and its Bio-PEPA specification. In Section 4.1.4, we present and validate our results. In Section 4.1.8, we present a summary of analysis of a robot swarm using Bio-PEPA.

4.1.1 Bio-PEPA

Bio-PEPA is a process algebra originally developed by Ciocchetta and Hillston (2009) based on the Performance Evaluation Process Algebra (PEPA) (Hillston, 1996). For more information about process algebras, see Section 2.2.1.

The particular aspect that distinguishes Bio-PEPA from other process algebras is the way in which processes are interpreted. In Bio-PEPA, processes represent groups of similar entities. The interactions between processes, that is, the interactions between groups of entities, affect their population sizes. In the context of swarm robotics we use the abstraction of "processes as groups of entities" to model groups of robots performing different operations. We use the concept of interaction to model, for example, the movement of robots between different locations and for the formation of teams. The idea is that movement of robots between locations can be modeled as a simultaneous decrease of the size of a population situated in the location that is left and the increase of the size of a population in the location of arrival. This is similar to models based on rate equations. Regarding team formation, another feature, specific to Bio-PEPA, is particularly useful: the possibility to express the multiplicity of entities involved in single interactions, known as "stoichiometry" in the context of biochemistry. With this feature one can specify, for example, that three single robots can form a single team that subsequently is treated as a single entity in a new kind of "species".

Interactions among processes have durations that are modeled as continuous random variables with negative exponential distributions. The use of negative exponential distributions stems form the fact that Bio-PEPA semantics is based on continuous time Markov chains (CTMCs). In practice, restriction to exponential distributions does not represent a real limitation since it has been shown that any random distribution can be approximated by suitable combinations of negative exponential ones, of course at the cost of larger models, which can be ameliorated by suitable exploitation of process algebra equivalences (Tschaikowski and Tribastone, 2012).

A further feature of Bio-PEPA is that the rates at which interactions occur can be defined as general functions of the sizes of the groups involved in the interaction. This provides great flexibility in the definition of such rates[1].

[1] At the current state of Bio-PEPA, some Bio-PEPA specifications cannot be analyzed with the full array of analysis methods available. For example, it is not possible to directly use Gillespie's stochastic simulation algorithms to analyze specifications with interactions that depend on more than two species as input. An intermediate step to simplify the model is necessary. Additionally, caution is needed with the interpretation

We now briefly present the aspects of the Bio-PEPA language that are directly relevant for this dissertation. The interested reader can find further details of Bio-PEPA in Ciocchetta and Hillston (2009).

Bio-PEPA specifications consist of two main kinds of components.

The first kind of components is called the *"species" component*. Each species defines the behavior of *all* the individuals belonging to it. Species components are composed together in order to build a model using the *parallel composition* operator with *synchronization* on shared actions. The name "species" derives from the biochemical origins of Bio-PEPA; in the context of swarm robotics, the name "population" is usually preferred; in the following, we will consider "species" and "population" as synonyms. Species components are usually called processes in other process algebras.

The second kind of component is called *model component*. Model components define how species components are composed together in order to build a model using the *parallel composition* operator with *synchronisation* on shared actions.

The syntax of Bio-PEPA components is thus defined as follows, where S stands for a *species component* and P for a *model component*:

$$S ::= (\alpha, \kappa)\ \mathsf{op}\ S \mid S + S \mid C \ \text{ with } \mathsf{op} = \downarrow \mid \uparrow \mid \oplus \mid \ominus \mid \odot \ \text{ and } P ::= P \underset{\mathcal{L}}{\bowtie} P \mid S(x)$$

The *prefix combinator* "op" in the prefix term (α, κ) op S represents the impact that action α has on species S. Specifically, $\downarrow$ indicates that the number of entities of species S decreases when α occurs, and $\uparrow$ indicates that this number increases. The amount of the change is defined by the (stoichiometry) coefficient κ. This coefficient captures the multiples of an entity involved in an interaction. The default value of κ is 1, in which case we simply write α instead of (α, κ). To better understand this formula consider the follow example: $(a, 3){\downarrow}B$, which can be interpreted as upon action a, three instances of species B are removed from the population. Action durations are assumed to be random variables with negative exponential distributions, characterized by their *rates*. The rate of action α is defined by a so called functional rate or kinetic rate. Action rates are defined in the context section of a Bio-PEPA specification. The operator "+" expresses the choice between possible actions, and the constant C is defined by the equation $C{=}S$.

The process $P \underset{\mathcal{L}}{\bowtie} Q$ denotes cooperation between model components P and Q,

of numerical solutions of the sets of ordinary differential equations derived from a Bio-PEPA specification. We will address these issues in more detail in the analysis of the robot swarm decision-making strategy in Section 4.1.4. More information can be found in Ciocchetta and Hillston (2009).

the set $\mathcal{L}$ determines those actions on which P and Q are forced to synchronize. The shorthand $P \underset{*}{\bowtie} Q$ denotes cooperation on all actions that P and Q have in common. In the model component $S(x)$, the parameter $x \in \mathbb{R}$ represents the initial amount of the species.

As a simple example, consider two groups of robots, R and B, identified by their respective color, red and blue. Assume that we want to model the formation of a group T of teams each composed of two red robots and a blue one. Assume furthermore that the team formation occurs with a rate r that is proportional to the population size of red and blue robots. In Bio-PEPA this behavior can be modeled as follows. The effect of the formation of a team, represented by action *mk_team*, on the three "species" can be defined as:

$$R \overset{def}{=} (mk\_team, 2){\downarrow}R \qquad B \overset{def}{=} (mk\_team, 1){\downarrow}B \qquad T \overset{def}{=} (mk\_team, 1){\uparrow}T$$

The system can then be described by the following model component:

$$(R(r_0) \underset{\{mk\_team\}}{\bowtie} B(b_0)) \underset{\{mk\_team\}}{\bowtie} T(0)$$

where r_0 and b_0 denote the initial population sizes of the groups of red and blue robots, respectively, and 0 denotes that initially T is empty. What remains to define is the rate at which the teams are formed, that is, the rate of action *mk_team*. This rate could be defined as $f_{mk\_team} = r \times R \times B$.

Operationally, what happens is that every time a *mk_team* action occurs, the three species that share this action are synchronized and two red robots and one blue are taken away from the species R and B, respectively. At the same time one new team is generated and added to the species T increasing its population size by 1. How often the *mk_team* action occurs is given by its rate function, which in turn depends on the actual population sizes of the species R and B and a constant rate r. Note also that, whenever one of these population sizes become zero, the rate goes to zero too and the interaction can no longer take place.

One of the distinguishing characteristics of Bio-PEPA are *locations*, which are meant to be a symbolic representation of physical space. Locations are specified by extending prefix terms with the notation @. For instance, in order to specify that action α, has an effect op on population S that is *located* in location l, and in particular involves κ individuals of S, we write (α, κ) op $S@l$. Additionally, locations are used in model components in order to specify the initial size of the various populations in each location. Each location used in a Bio-PEPA specification must be declared; thus, a Bio-PEPA *system* specification with *loca-*

tions consists of a set of species components, a model component, and a context containing definitions of locations, functional/kinetic rates, parameters, and so on.

Bio-PEPA is given a formal operational semantics based on continuous time Markov chains (CTMCs) and on ordinary differential equations (ODE) (Ciocchetta and Hillston (2008) and Ciocchetta and Hillston (2009)).

Bio-PEPA is supported by a suite of software tools which automatically process Bio-PEPA models and generate internal representations suitable for different types of analysis as described in detail in Ciocchetta and Hillston (2009) and Ciocchetta et al. (2009). These tools include mappings from Bio-PEPA to differential equations (ODE) supporting a fluid flow approximation (Hillston, 2005), stochastic simulation models (Gillespie, 1977), CTMCs with levels (Ciocchetta and Hillston, 2008) and PRISM models (Kwiatkowska et al., 2011) amenable to statistical model checking. Consistency of the analyses is supported by a rich theory including process algebra, and the relationships between CTMCs and ODE.

4.1.2 Collective decision-making: a Bio-PEPA specification

To demonstrate the characteristics of Bio-PEPA, in this section we analyze a robot swarm performing collective decision-making, as originally presented by Montes de Oca et al. (2011)[2]. The goal of the robot swarm is to perform foraging: the robots carry objects from a *start* area to a *goal* area. Differently from other foraging scenarios (Brambilla et al., 2013), the objects to be carried are too heavy for a single robot, thus cooperation is necessary: the robots form teams of three to be able to carry an object.

The scenario is very similar to the ants double bridge experiment (Goss et al., 1989). The start and the goal areas are connected by two paths: a short path and a long path. Similar to what ants do in the double bridge experiments, robots have to collectively identify and choose the shortest path. Differently from what ants do, robots do not use pheromones but a voting process based on the majority rule.

Each robot has a preferred path. When a group is formed in the start area (Figure 4.1a), a vote takes place and the group chooses the path that is preferred by the majority of the robots composing it (Figure 4.1b). The chosen path also

[2]Since an implementation of this system using real robots is not available, the physics-based simulation will be considered our *ground truth*, that is, not another analysis phase, but the subject of our analysis effort.

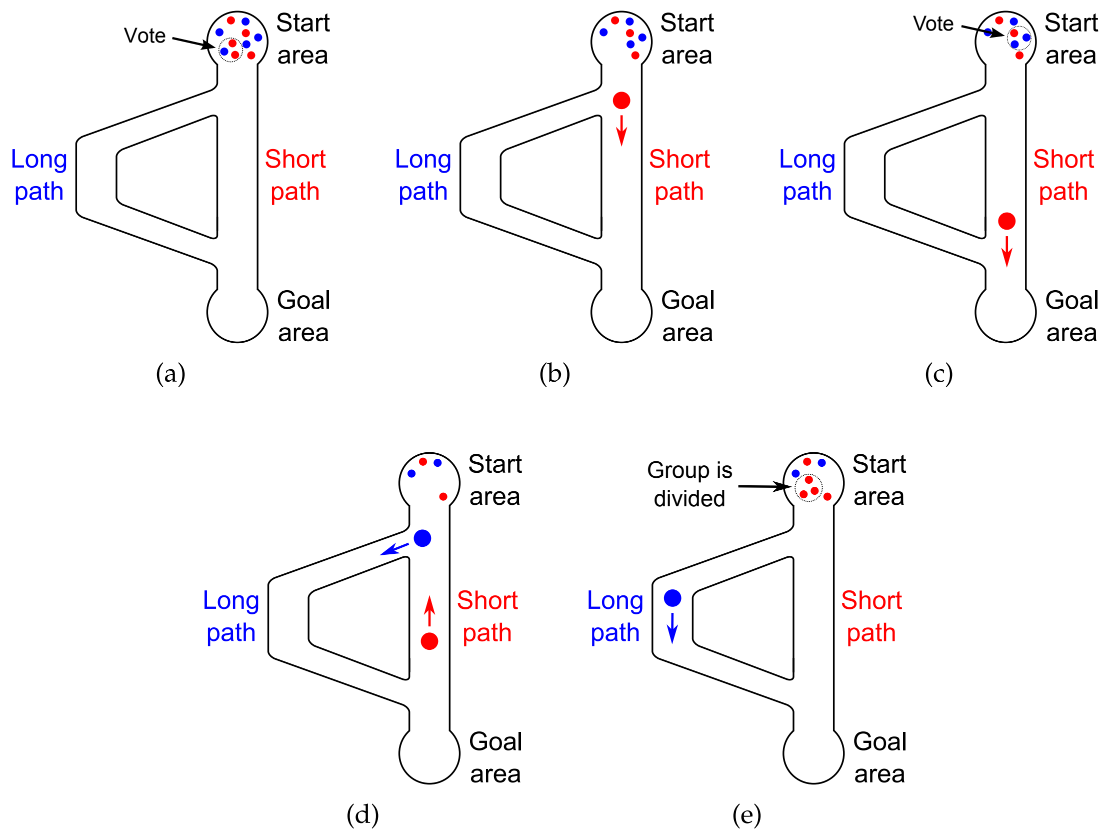


Figure 4.1: The foraging scenario analyzed in this section. The robots start in the start area. Groups are formed and their path is chosen using the majority rule. In this figure two examples of the voting process are shown: (a) a group is formed; (b) the group has chosen the short path; (c) while the first group is active, another group is formed; (d) the second group has chosen the long path, at the same time the first group is coming back; (e) the first group is back in the start area and is disbanded; note how all the robots of the disbanded group have now the same preference.

becomes the new preferred path for all the robots composing the group (Figure 4.1e). For example, if two robots prefer the short path and one robot prefers the long path, the short path is chosen for the next run, and the robot that preferred the long path changes its preference to the short path. Note that the voting process takes place only in the start area and no other event can change the preference of the robots. This means that robots come back to the start area following the same path taken for the outgoing trip. Figure 4.1 shows a schema of the scenario.

Since the robots taking the short path spend less time out of the start area than the robots taking the long path, their participation in the vote is, on average, more frequent. This results in the formation of more groups preferring the short path. If, initially, half of the robots have a preference for the short path and half for the long path, over time, all robots will converge on preferring the short path. More details are given in the work by Montes de Oca et al. (2011) and in Section 4.1.3.

We chose this collective decision-making behavior as a case study for Bio-PEPA since it displays several interesting characteristics common to many robot swarms:

- Simplicity: the collective behavior is simple enough that it is possible to analyze it without being hampered by the implementation details.
- Direct cooperation: the robots must form groups of three to carry the objects.
- Indirect cooperation: the vote process creates an opinion dynamic that let the robots collectively choose the shortest path.
- Space and time aspects: space and time play an important role and must be carefully modeled. In particular, the voting process is spatially located in the start area and only the robot in the start area at a given moment can take part in it. Additionally, the time necessary for the robots to carry an object, which depends on the length of the chosen path, affects the opinion dynamic.

This system has been analyzed in several other works: Montes de Oca et al. (2011) presented a simple fluid flow analysis and a Monte Carlo simulation, Scheidler (2011) presented a more complex fluid flow analysis, Valentini et al. (2012) presented an analysis based on absorbing Markov chains. In these works, each analysis was based on a different model. In our work, we use a single

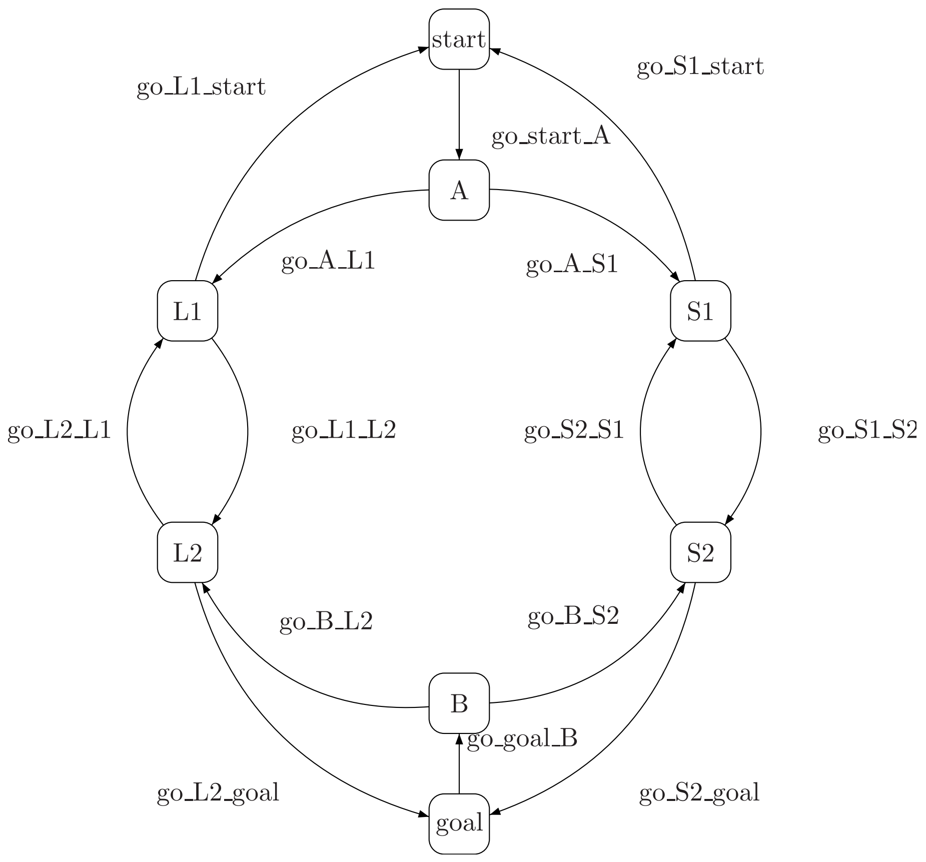


Figure 4.2: Locations and transitions of robots in the simplified Bio-PEPA specification.

Bio-PEPA description to perform three different kinds of analysis: stochastic simulation, model checking and fluid flow analysis.

4.1.3 The Bio-PEPA specification

In this section, we present the Bio-PEPA specification of the system.

As shown in Figure 4.2, the system is described through eight Bio-PEPA *locations*: two boundary locations, `start` and `goal`; a location *A* where robot teams select the short or long path to `goal` according to the decision taken when leaving `start` and, similarly, location *B*, where robot teams select the short or long path back to `start`, again according to the previously taken decision. We have then two locations for each path, *L1* and *L2* for the long path and *S1* and *S2* for the short one.

We also define a set of Bio-PEPA *species* to specify the behavior of the robots. For example in `start` we distinguish two species of robots: those that the last time returned via the short path, denoted as *Robo_start_fromS*, and those that

$$
\begin{aligned}
\mathit{Robo\_start\_fromS} \quad = \quad & (\mathit{allS}, 3)\downarrow \mathit{Robo\_start\_fromS@start} + \\
& (\mathit{S2L1}, 2)\downarrow \mathit{Robo\_start\_fromS@start} + \\
& (\mathit{S1L2}, 1)\downarrow \mathit{Robo\_start\_fromS@start} + \\
& (\mathit{go\_S1\_start}, 3)\uparrow \mathit{Robo\_start\_fromS@start}; \\
\\
\mathit{Teams\_A\_S} \quad = \quad & (\mathit{allS}, 1)\uparrow \mathit{Teams\_A\_S@A} + \\
& (\mathit{S2L1}, 1)\uparrow \mathit{Teams\_A\_S@A} + \\
& \mathit{go\_A\_S1}\downarrow \mathit{Teams\_A\_S@A};
\end{aligned}
$$

Figure 4.3: Two species components synchronized on action *S2L1*.

returned via the long path, denoted as *Robo_start_fromL*. In the following we will refer to these two groups also as the *S-population* and the *L-population*, respectively. Similarly, other locations contain populations of teams of robots that move in the direction from the start area to the goal area and those that move in the opposite direction. For example, in location *S1* we can have *Teams_S1_StoG* and *Teams_S1_GtoS*, where *StoG* denotes the direction from the start area to the goal area and *GtoS* the opposite direction.

The Bio-PEPA fragment below specifies the behavior of a robot. Robots leave the start area in groups of three. Each group is randomly composed of three robots. There are four possible compositions:

- all robots from the S-population;
- all robots from the L-population;
- two robots from the S-population and one robot from the L-population;
- two robots from the L-population and one robot from the S-population.

These combinations are modeled as four different actions: *allS*, *allL*, *S2L1* and *S1L2*. In Bio-PEPA, the formation of teams of robots is modeled by the coefficient that indicates how many entities are involved in an action. For example, upon action *allS*, three robots of the S-population leave `start` (indicated by $(\mathit{allS}, 3)\downarrow$), to form a *team* in choice point *A* (indicated by $(\mathit{allS}, 1)\uparrow$ in *Teams_A_S*) which will take the short path when it continues its journey towards the goal area (population *Teams_A_S@A*). Since action *allS* is shared between the species components *Robo_start_fromS* and *Teams_A_S*, this movement occurs simultaneously with the rate of action *allS* that will be defined later on.

In a similar way, upon action *S2L1*, which is present in the three species components *Robo_start_fromS*, *Teams_A_S* and *Robo_start_fromL* (two of which are shown in Figure 4.3), all three species components synchronize, resulting in two robots from the S-population and one from the L-population leaving the start area and forming at the same time one new team in choice point A in the population *Teams_A_S*. *Teams_A_S* is the population of those teams in choice point A that decided to take the short path. The synchronization pattern of the components is given by the model component shown later on. The excerpt above only shows the behavior of teams voting for the short path. The behavior of those voting for the long path is similar and omitted for reasons of space. For the same reason also the behavior of teams moving between different locations is not shown.

The actions denoting teams of robots leaving the start area need to occur with appropriate rates. In particular, it is defined as a fixed rate *move*, multiplied by the probability of forming a team with a particular composition:

- A group of three robots that are all from the S-population has a probability to occur equal to

  $$pSSS = \frac{(RSS)}{(RSS)+(RSL)} \cdot \frac{(RSS-1)}{(RSS-1)+(RSL)} \cdot \frac{(RSS-2)}{(RSS-2)+(RSL)}$$

  where, for the sake of readability, *Robo_start_fromL@start*, which is the population of robots with preference for the long path in the start area, is abbreviated as RSL and *Robo_start_fromS@start*, which is the dual population with preference for the short path, is abbreviated as RSS.

  Table 4.1 lists all species components used in the Bio-PEPA specification.

- A similar probability $pLLL$ can be defined for a group of three robots from the L-population.

- The probability to extract two robots from the S-population and one from the L-population is:

  $$pSSL = \frac{(RSS)}{(RSS)+(RSL)} \cdot \frac{(RSS-1)}{(RSS-1)+(RSL)} \cdot \frac{(RSL)}{(RSS-2)+(RSL)}$$

- Similarly probabilities for $pSLS$, $pLSS$, $pLLS$, $pLSL$ and $pSLL$ can be defined.

- Therefore, the total probability that two, out of the three members of a team, vote for the short path is $pSSL + pSLS + pLSS$ while that for the long path is $pSLL + pLSL + pLLS$.

Once these probabilities are defined, we can define the rates of the related events: The rates of actions *S2L1* and *S1L2* can now be defined as $(pSSL + pSLS + pLSS) \cdot$

move and $(pSLL + pLSL + pLLS) \cdot move$, respectively. Note that the sum of these eight probabilities, *pSSL*, *pSLS*, *pLSS*, *pSLL*, *pLSL*, *pLLS*, *pSSS* and *pLLL*, amounts to 1. In this way, the total rate at which teams of robots leave the start area is constant and given by the parameter 'move' independently from the composition of the teams formed.

The rate at which teams move from *A* to *S1* and to *L1* is dependent on the number of teams present in *A* and are $walk\_normal \cdot Teams\_A\_S@A$ and $walk\_normal \cdot Teams\_A\_L@A$, respectively. The rate parameter *walk_normal* specifies the time it takes a robot team to move from choice-point *A* to the first section of a path.

Finally, the overall system definition shows the initial size of robot populations in each location. The overall robot behavior is defined using cooperation on shared actions (see page 91 for a definition and example):

$$\begin{aligned}
&Robo\_start\_fromS@start(SS) \underset{*}{\bowtie} Robo\_start\_fromL@start(SL) \underset{*}{\bowtie}\\
&Teams\_A\_S@A(0) \underset{*}{\bowtie} Teams\_A\_L@A(0) \underset{*}{\bowtie}\\
&Teams\_S1\_StoG@S1(0) \underset{*}{\bowtie} Teams\_S1\_GtoS@S1(0) \underset{*}{\bowtie}\\
&Teams\_S2\_StoG@S2(0) \underset{*}{\bowtie} Teams\_S2\_GtoS@S2(0) \underset{*}{\bowtie}\\
&Teams\_L1\_StoG@L1(0) \underset{*}{\bowtie} Teams\_L1\_GtoS@L1(0) \underset{*}{\bowtie}\\
&Teams\_L2\_StoG@L2(0) \underset{*}{\bowtie} Teams\_L2\_GtoS@L2(0) \underset{*}{\bowtie}\\
&Teams\_goal\_fromS@goal(0) \underset{*}{\bowtie} Teams\_goal\_fromL@goal(0) \underset{*}{\bowtie}\\
&Teams\_B\_fromS@B(0) \underset{*}{\bowtie} Teams\_B\_fromL@B(0)
\end{aligned}$$

where the number *SS* in $Robo\_start\_fromS@start(SS)$ (resp. *SL*) is the initial size of the robot S-population (resp. L-population) present in the start area (*@start*).

There is a further issue to consider: how to model the length of the paths. This can be done in two ways. The first is to model each path by two sections, as illustrated above, and set one path longer than the other by choosing a different rate for the movement between sections on the paths. However, as also discussed in Montes de Oca et al. (2011), this model has the disadvantage that the duration of path traversal is essentially modeled by a *short* series of exponential distributions which in general approximates the average duration well, but not the variability. It therefore does not reflect very well realistic robot behavior. An alternative is to choose the same rate for each section and to vary the number of sections on each path to model their difference in length. In this way, the traversal time of a path is modeled by a sequence of say *m* exponentially distributed random variables with rate λ, also known as an Erlang distribution, using the well-known method of stages (see Kleinrock (1975) p. 119).[3]

[3] The mean (variance, resp.) of an Erlang distribution with m phases of rate λ is m/λ (m/λ^2 resp.).

We model the two paths of the environment with 8 sections for the short path and 15 sections for the long path. Each section takes, on average, ten time units to traverse by a robot team. This is modeled in the system by defining the rate $walk\_normal = 0.1$. Considering also the movements from the choice points to the path and those from the path to the start area and the goal area, in this way the short path takes on average 100 time units to traverse, and the long one 170. This is comparable to the latency periods used in Montes de Oca et al. (2011) (end of Section 4) and provides a good approximation of the actual variability observed in robot movement. Other free variables of the model not provided in Montes de Oca et al. (2011) have been selected by us.

The analysis presented in Montes de Oca et al. (2011), which we will use to compare our results with in the next section, is based on the assumption that, at any time, there is a constant number k of active teams (that is, not in the start area). The number k is a parameter of the model. To model this in the Bio-PEPA model, we use the parameter min_start, which is a rate that can be used to control the minimum number of robots in the start area at any time. As we will see in the next section, after a short initial transitory period, the following holds in the model with a good approximation: $k = (32 - min\_start)/3$.[4]

A more detailed graphical representation of the complete Bio-PEPA specification is presented in Figure 4.4. The figure presents the various locations of the model. In each location there are two populations. Their names have been abbreviated for reasons of presentation. Names starting by R indicate populations of robots, names starting by T refer to populations of teams. Names ending in S refer to populations of elements that are in favor of the short path, those ending in L refer to elements in favor of the long path. The arrows after the names of the populations in the locations on the paths indicate the direction of movement of the elements of the population, so those moving from the start area to the goal area are indicated by an arrow pointing downwards, whereas those moving from the goal to the start are indicated by an arrow pointing upwards. The actions that label the transitions between locations correspond to those in the Bio-PEPA specification, though, for reasons of readability, only one action is shown ($allS$) of all those between the start area and choice point A.

Thus an appropriate choice of m and λ can guarantee the required values for the mean and variance, approximating a normal distribution.

[4]In Bio-PEPA, one can make use of a predefined function H which takes a number as an argument. If this number is zero, H returns zero, otherwise it returns 1. To guarantee a minimum number min_start of robots, in the start area, the rate of action $S2L1$ can then be defined as: $S2L1 = (pSSL + pSLS + pLSS) * move * H((RSS + RSL) - min\_start)$; the same must be done for the other related rates.

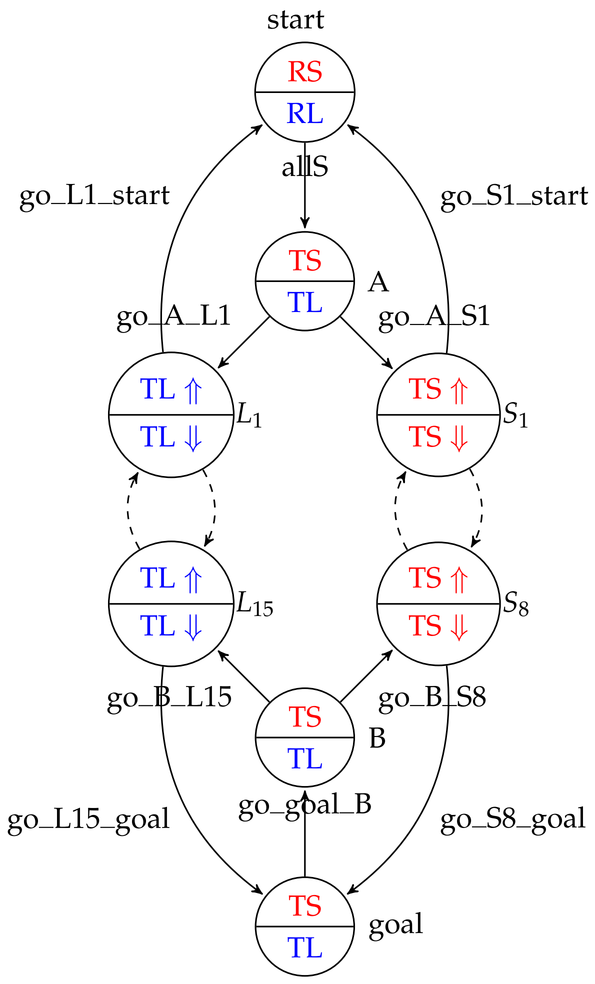


Figure 4.4: Graphical representation of the full Bio-PEPA swarm decision-making model. Note that, for reason of clarity, we do not show the 8 locations on the short path (S_1 through S_8) nor the 15 on the long path (L_1 through L_{15}), but only the first and last. Arrows ⇑ and ⇓ indicate the moving direction of the related teams.

Table 4.1: All species components used in the Bio-PEPA specification.

Species name	Description
Robo_start_fromS	Robots in location *start* which prefer the short path.
Robo_start_fromL	Same as above, but for the long path.
Teams_A_S	Teams at location *A* , that is, performing the choice between the two paths, which prefer the short path.
Teams_A_L	Same as above, but for the long path.
Teams_S1_StoG ... *Teams_S8_StoG*	Teams on the short path, which is composed of a total of 8 states, going from *start* to *goal*.
Teams_S1_GtoS ... *Teams_S8_GtoS*	As above, but from *goal* to *start*.
Teams_L1_StoG ... *Teams_L15_StoG*	Teams on the long path, which is composed of a total of 15 states, going from *start* to *goal*.
Teams_L1_GtoS ... *Teams_L15_GtoS*	As above, but from *goal* to *start*.
Teams_B_fromS	Teams at location *B* coming from the short path.
Teams_B_fromL	Same as above, but from the long path.
Teams_goal_fromS	Teams at location *goal* from the short path.
Teams_goal_fromL	Same as above, but from the long path.

4.1.4 Analysis

In the following, we illustrate three different forms of analysis of the same Bio-PEPA specification and compare their results with those available in the literature (Montes de Oca et al., 2011). Good correspondence of the results would validate our model and confirm that Bio-PEPA is a viable formal language to analyze a robot swarms, with the additional advantage that a single specification can be used for multiple kinds of analysis. This also means that, due to the precise and unambiguous mathematical semantics of the language, the results of the different analyses are formally related and coherent since they are systematically derived from the same specification.

We consider a swarm with a population of 32 robots, unless stated otherwise. We furthermore consider the following parameters for the model: initially $SS = 16$ and $SL = 16$, $move = 0.28$, $walk\_normal = 0.1$.

4.1.5 Stochastic simulation

The first kind of analysis we present uses stochastic simulation. The analysis we perform using stochastic simulation is limited in its depth. Our goal here is not to perform a complete analysis of the system using stochastic simulation, but to demonstrate its feasibility and to provide minimal results that will be used as starting point for a more in depth analysis performed using other approaches.

We use stochastic simulation to check the average number of active teams for different assumptions on the minimal number of robots that are present in the start area. This validation is necessary to be able to compare the results of our analysis with those presented in Montes de Oca et al. (2011).

The Bio-PEPA tool suite relies on an implementation of Gillespie's stochastic simulation algorithm (Gillespie, 1977). The original algorithm assumed that only interactions with at most two species were used in the model and that the rates were simple products of a constant and a population size. In the Bio-PEPA model, this is indeed the case with the exception of the rate functions involved in the team formations, which are slightly more general, but can be reduced to a product of a constant and the population size.

Figure 4.5 presents two stochastic simulation results, averaged over 10 simulation runs, for $min\_start = 5$ (Figure 4.5 left) and $min\_start = 2$ (Figure 4.5 right), showing the number of robots on both paths and in the start area, and the number of teams on each path. The figure shows that the number of active teams on the paths quickly increases to 9 in case of $min\_start = 5$ and 10 in case of *min_start* $= 1$ and then stabilizes. This means that the rate at which robots leave the start area, i.e. *move* $= 0.28$, is sufficiently high to quickly reach a situation that presents the desired number of active teams. This allow us to compare the results of this analysis with the results obtained with the physics-based simulation and Monte Carlo simulation reported in Montes de Oca et al. (2011).

4.1.6 Statistical model checking

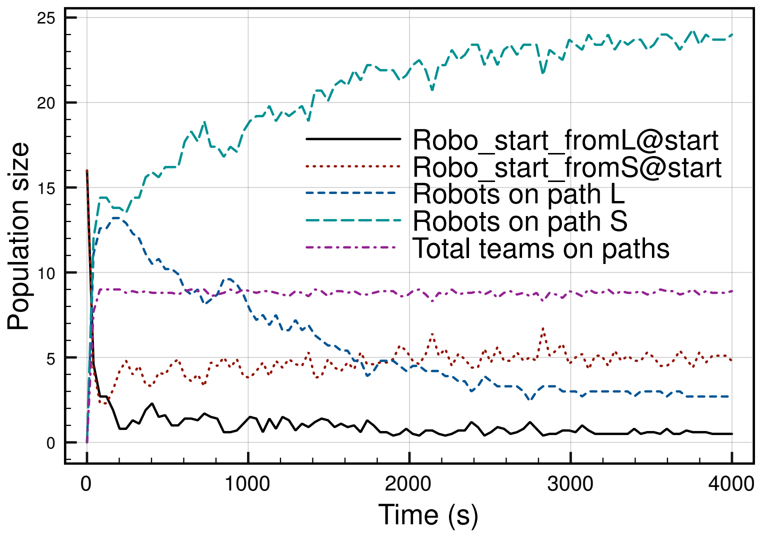
For the analysis of properties of the Bio-PEPA model we will make use of *statistical model checking*, as the number of states of the model makes complete model checking computationally intractable. In particular, we use confidence interval methods, as presented in Section 2.2.3, to estimate probability and rewards.

The Bio-PEPA specification developed in Section 4.1.2 can be translated automatically into a model expressed in the PRISM input language by the Bio-PEPA tool suite described in Ciocchetta et al. (2009). The translation mechanism itself is described in Ciocchetta and Hillston (2009). The PRISM model is a stochastic model having a CTMC as underlying mathematical structure.

One of the principal properties of interest for decision making involves convergence. The first concern is whether convergence on one of the paths occurs at all. In principle, the swarm could converge to a mixed decision situations, in which no single path is chosen. We will show that such situation occurs with zero probability. A second concern is whether convergence on a single path always occurs eventually, that is, the system does not enter in some form of oscillation that prevents convergence. Two kinds of convergence would be possible: convergence on the long path and convergence on the short path. Convergence on the short path (*Convergence_on_S*) can be defined as the situation in which each of the 32 robots is either in a team on the short path, or in the S-population in the choice points, the start area or the goal area.

In terms of the population sizes in the various locations, convergence on the short path can be formalized as the following proposition:

$$\begin{aligned} Convergence\_on\_S \equiv\, & 3 * (Teams\_S1\_StoG@S1 + \cdots + Teams\_S8\_StoG@S8) + \\ & 3 * (Teams\_S1\_GtoS@S1 + \cdots + Teams\_S8\_GtoS@S8) + \\ & 3 * Teams\_goal\_fromS@goal + Robo\_start\_fromS@start + \\ & 3 * (Teams\_A\_S@A + Teams\_B\_fromS@B) = 32 \end{aligned}$$



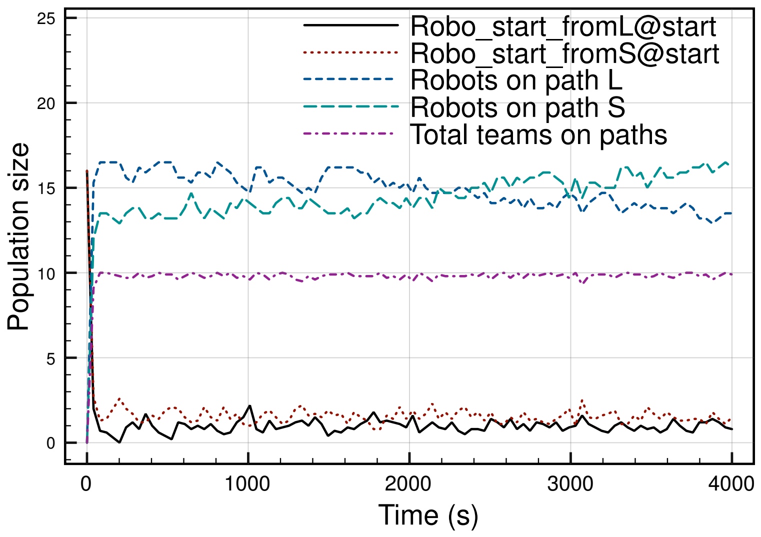


Figure 4.5: Number of active teams for *min_start* = 5 (left) and *min_start* = 2 (right) with *move* = 0.28. The graphs show the average values over 10 runs.

"Convergence_on_L" can be defined similarly, but requiring that the above sum is equal to 0 instead of 32.

The formula to obtain an estimate of the probability that the system eventually converges either on the long or on the short path can now be expressed in terms of the formulae just introduced:

$$P = ?\,[true\ U\ (\text{“}Convergence\_on\_L\text{”}\ \mid\ \text{“}Convergence\_on\_S\text{”})] \tag{4.1}$$

Recall that $P = ?$ is used to compute a probability, and U reads as "until".

For 100 sample paths, a confidence level $\alpha = 0.01$ and a maximum sample path length of 20,000 we obtain that for each k ranging from 1 to 10 the system converges to the short or the long path with probability 1. In fact, convergence takes place in each of the sample paths, so mixed decision situations do not occur.

The next question of interest is then what is the probability that the system converges on the short path. More precisely, this question should be formulated as "what is the probability that the system did not converge on the long path until it converges on the short path". The latter can be expressed as:

$$P = ?\,[!\text{“}Convergence\_on\_L\text{”}\ U\ \text{“}Convergence\_on\_S\text{”}] \tag{4.2}$$

where that ! stands for negation.

The analyses of Formula 4.2 for a number of teams k ranging from 1 to 10 is shown in Figure 4.6 as a solid line. The analyses have been based on 100 random sample paths, a confidence level $\alpha = 0.01$ and a maximal sample path length of 20,000. In the figure the widths of the confidence interval are shown as vertical bars. The results are compared to those obtained via physics-based simulation and Monte Carlo simulation of the same case study reported in Montes de Oca et al. (2011) and shown as dotted and dashed lines, respectively. The latter are close to the results obtained with the Bio-PEPA specification and well within the error-margins.

The expected number of teams formed until convergence has taken place on the short or the long path can be analyzed by statistical model checking using a *reward formula*:

$$R\{\text{“}teams\text{”}\} = ?\,[F\ (\text{“}\_Convergence\_on\_S\text{”}|\text{“}\_Convergence\_on\_L\text{”})] \tag{4.3}$$

The formula refers to the reward structure labeled "teams" that counts the num-

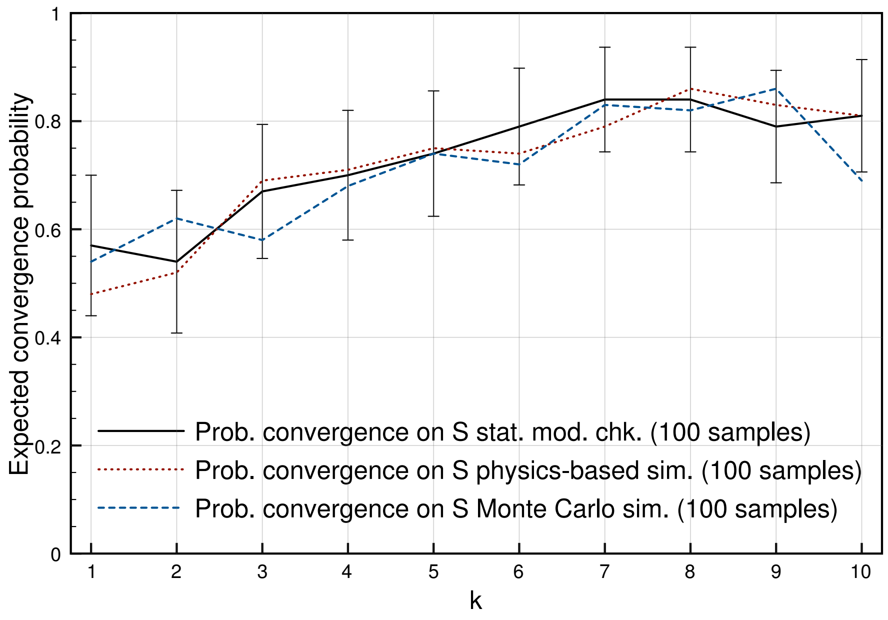


Figure 4.6: Probability of convergence on the short path (100 samples). k is the number of active teams in the system.

```
reward  ‘‘teams’’
  [go_A_S1] true :  1;
  [go_A_L1] true :  1;
 endreward
```

Figure 4.7: *Reward structure* to count team formations.

ber of teams that were formed. In terms of the Bio-PEPA model, the formation of teams is directly related to the occurrence of the actions ‘go_A_S1’ and ‘go_A_L1’, that is, when teams move from choice point A to one of the paths. The specific *reward structure* required is shown in Figure 4.7. Essentially this represents the fact that every time action ‘go_A_S1’ or ‘go_A_L1’ occurs, the total number of teams formed so far is incremented by 1.

Figure 4.8 shows results on the expected number of team formations until convergence on the short or long path (Formula 4.3) using 1000 samples, $\alpha = 0.01$ and maximal path length of 20,000. The width of the confidence intervals are shown as error-bars.

The results obtained by statistical model checking, physics-based simulation and Monte Carlo simulation are consistent for values of k up to 7. They diverge for higher values of k. The divergence can be explained by the differences in

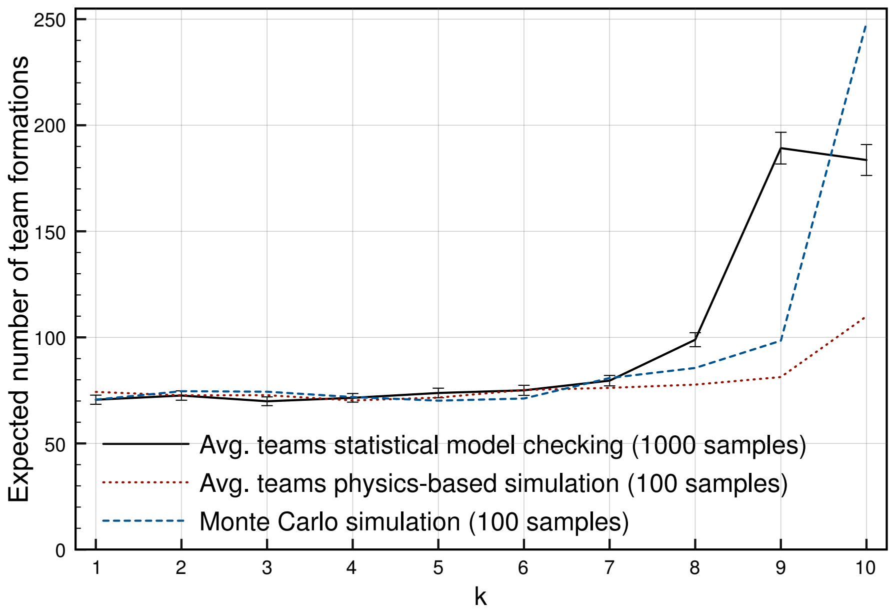


Figure 4.8: Expected number of team formations until convergence (1000 samples). k is the number of active teams in the system.

the underlying models that are used. The Monte Carlo simulations are obtained from an ODE model in which it is assumed that, at any point in time, a constant fixed fraction of the total population is in the start area. Such a fixed fraction can only be maintained if, upon arrival of a team in the start area, a new team forms and leaves the start area immediately. In the Bio-PEPA model this can be approximated by choosing a high rate for the parameter 'move'. In fact, as can be observed in Figure 4.9, for $move = 30$ the results of the Bio-PEPA model follow a similar tendency as the results for the Monte Carlo simulation. An explanation for this tendency is that, for high values of k, the system needs more team formations to converge. This is due to the fact that when k is high, a robot team returning to the start area can influence the opinion only of the few robots that are in the start area: 5 robots for $k = 9$ and only 2 for $k = 10$.

For $k = 9$ there is a further divergence between the results obtained by Monte Carlo simulation and stochastic model checking. This can most likely be explained by the fact that Monte Carlo simulations start from an initial state in which a large fixed fraction of the population is already out of the start area and distributed over the paths in a particular proportion. The number of team formations needed to reach such a state is not considered in the Monte Carlo simulation. On the other hand, in the Bio-PEPA model (and in the physics-based

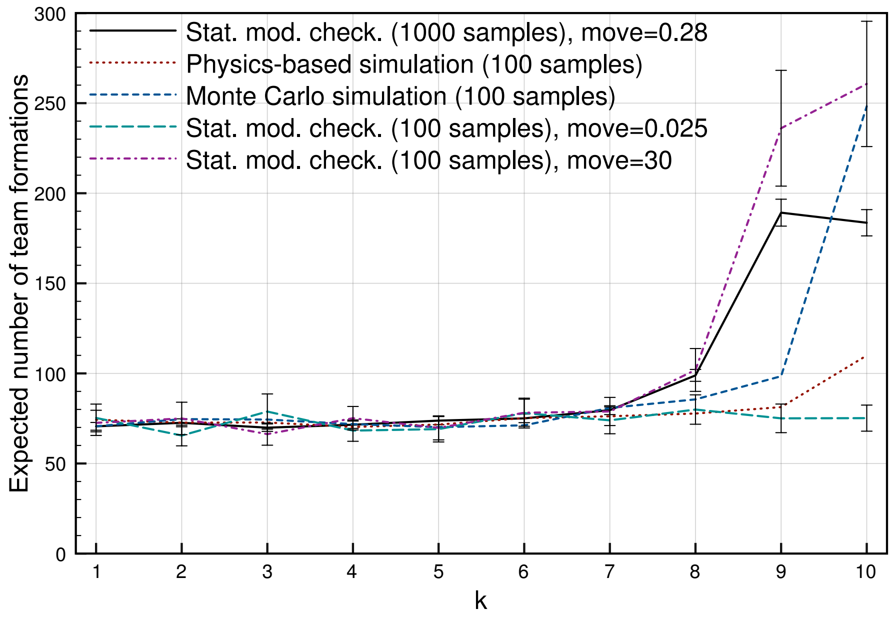


Figure 4.9: Expected number of team formations until convergence for different rates at which teams leave the start area in the Bio-PEPA model. k is the number of active teams in the system.

simulation) all robots are initially in the start area and subsequently distribute over the two paths. This results in many different intermediate distributions over the paths, which are likely to have an effect on the average number of team formations needed to reach convergence. Furthermore, for $k = 10$,and for $k = 9$ to a somewhat lesser extent, border effects might arise: the system is stretched to an extreme situation in which, at any time, only 2 robots remain in the start area. This small number is a source of strong stochastic fluctuations that might cause "accidental" convergence earlier than what one could expect given the size of the population.

The physics-based simulation is based on the assumption that the teams leave the start area on average every 40 seconds, until a number of k teams are active. In the Bio-PEPA specification, this can be modeled by setting $move = 0.025$. The formation of teams is suspended whenever there are k teams active and is resumed when teams return to the start area. For this value of $move$, statistical model checking produces results that are comparable with those produced by the physics-based simulation (as shown in Figure 4.9). These results can be explained by observing that in the model used for the physics-based simulation when k is high, the average number of active teams is actually substantially lower

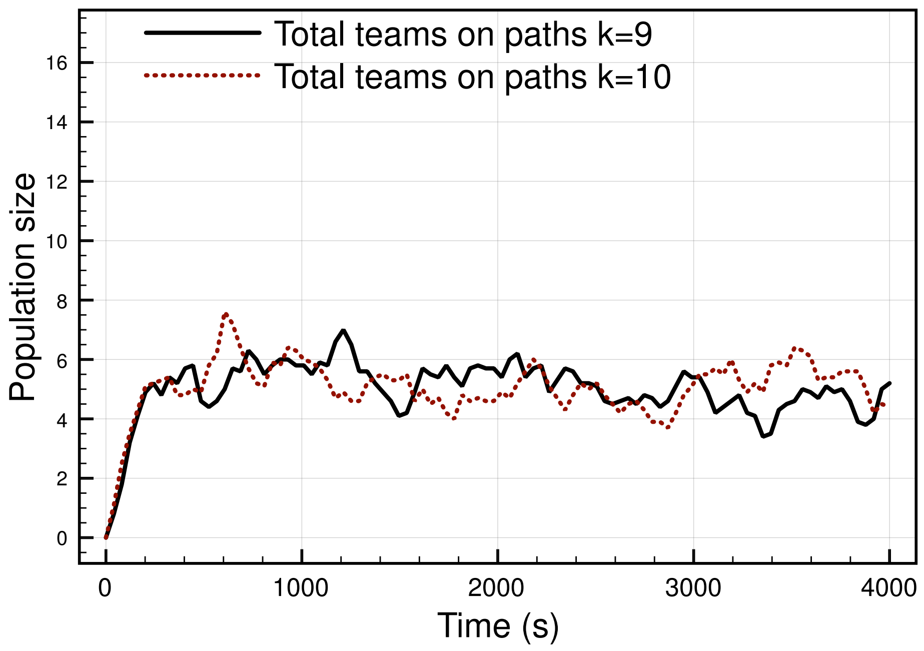


Figure 4.10: Number of active teams for *min_start* $= 5$ ($k = 9$) and *min_start* $= 2$ ($k = 10$) for *move* $= 0.025$ (average over 10 independent simulation runs).

than the nominal value k. This can also be made visible using simulation of the Bio-PEPA specification as shown in Figure 4.10 for an average of the number of active teams over 10 simulation runs for $k = 9$ and $k = 10$ and *move* $= 0.025$. As a consequence, the number of robots in the start area is larger than the nominal $N - 3k$, which in turn means that there are more robots that provide implicitly feedback on which of the two paths is the shortest. This explains why the expected number of teams formed until convergence obtained with statistical model checking does not differ much from those obtained with physics-based simulation (for *move* $= 0.025$).

The difference between physics-based simulation and statistic model checking for higher values of the parameter *move* can be explained by looking at the early phases of the experimental runs. In the early phases, there are more robots in the start area and they leave that area relatively quickly before feedback from returning teams can be taken into account. This is possibly leading to larger stochastic fluctuations before the system converges on one of the paths, resulting in more team formations.

A similar analysis using the same formula, but substituting *teams* with the reward structure *total_time* (shown in Figure 4.7), gives the expected time until

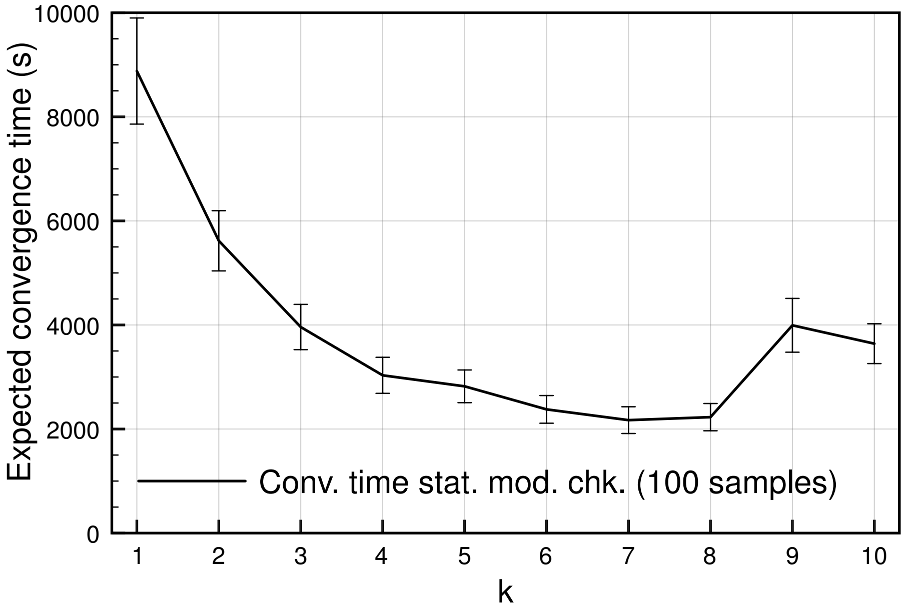


Figure 4.11: Expected convergence time (100 samples), *move* = 0.28.

convergence (for *move* = 0.28). Figure 4.11 shows the expected convergence time. No data from the literature concerning this aspect is available for comparison.

The total model-checking time to produce the data in Figure 4.6 was ca. 10 minutes, those in Figure 4.8 ca. 48 minutes and those in Figure 4.11 ca. 5 minutes[5].

By separating the *reward structure* in Figure 4.7 into one for the expected number of teams that decide to take the short path (S-teams) and one for those that decide to take the long path (L-teams), the contribution of each kind can be made visible using a *reward formula* similar to that shown in Formula (4.3). The result is shown in Figure 4.12. For any value of k the number of S-teams is always higher than the number of L-teams. This can be explained by the fact that initially the S-population and the L-population in the start area have equal size and that the probability that the system converges on the short path is more than 50% in all cases.

4.1.7 Fluid flow analysis

The third kind of analysis we consider is a fluid flow approximation of the ordinary differential equations (ODE) underlying the Bio-PEPA specification. Based

[5]Model-checking was performed on an iMAC with a 3.2 GHz Intel core i3 processor and 4 GB memory running the MacOS X operating system.

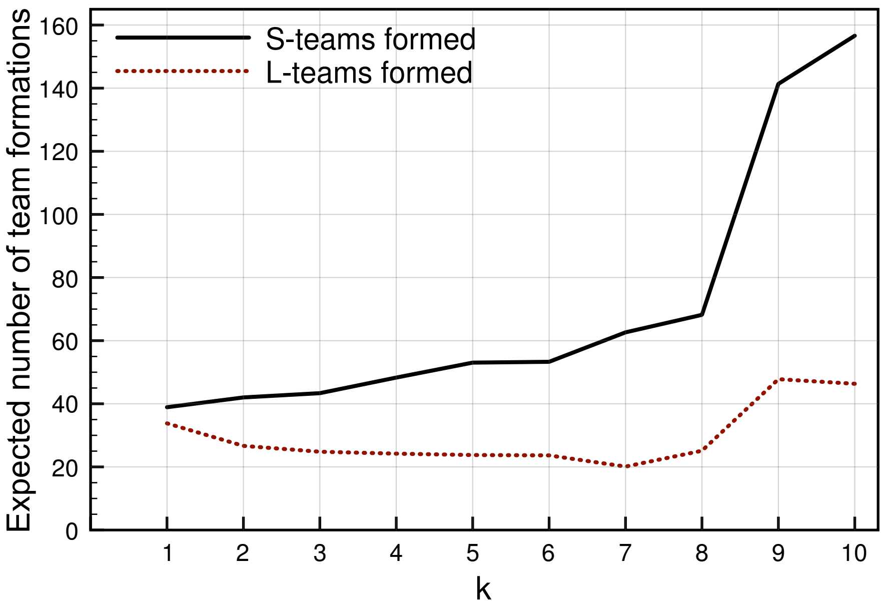


Figure 4.12: Expected S-teams and L-teams formed until convergence ($move = 0.28$).

on the Bio-PEPA syntax, the underlying ODE model can be generated automatically and in a systematic way, as shown in Hillston (2005) and in Ciocchetta and Hillston (2009), using the Bio-PEPA tool suite presented in Ciocchetta et al. (2009). This provides yet another view on the behavioral aspects of the system. It is then possible, for example, to explore numerically the sensitivity of the system to initial values and discover stationary points and other aspects related to stability analysis.

The derivation of an ODE model from a Bio-PEPA specification is based on the following steps (see Ciocchetta and Hillston (2009)):

1. define the $n \times m$ matrix D, where n is the number of species and m is the number of actions. The entries of the matrix D are obtained in the following way. For each species component C_i, the prefix sub-terms C_{ij} are considered. These are the sub-terms in the form (α_j, κ_{ij}) op $S_i@l$ defined in the Bio-PEPA model. Such sub-terms represent the change of the species i as a consequence of action j. If the term contributes to an increase of the population size of the species then the entry is $+\kappa_{ij}$, if it contributes to a decrease then the entry is $-\kappa_{ij}$;

2. define the $m \times 1$ vector $\mathbf{v}_f(t)$, where m are the rates of the action defined in the Bio-PEPA model;

3. associate the variable $x_i(t)$, the expected value of the population size at time t, with each component C_i and the definition of the $n \times 1$ vector $\mathbf{x}(t)$.

The ODE system is then obtained as:

$$\frac{d\mathbf{x}(t)}{dt} = D \times \mathbf{v}_f(t)$$

with initial population sizes x_{i_0}, for $i = 1, ..., n$.

To illustrate these steps, consider the slightly extended toy example introduced in Section 4.1.1 in which teams can also be dissolved into individual red and blue robots as follows:

$$\begin{aligned}
R &\overset{def}{=} (mk\_team, 2){\downarrow}R + (dis, 2){\uparrow}R \\
B &\overset{def}{=} (mk\_team, 1){\downarrow}B + (dis, 1){\uparrow}B \\
T &\overset{def}{=} (mk\_team, 1){\uparrow}T + (dis, 1){\downarrow}T
\end{aligned}$$

with the following model component:

$$(R(r_0) \underset{\{mk\_team,dis\}}{\bowtie} B(b_0)) \underset{\{mk\_team,dis\}}{\bowtie} T(t_0)$$

If we let the functional rates for this toy example be $mk\_team = 0.002 * R * B$ and $dis = 0.2 * T$ we obtain the following ODE:

$$\begin{aligned}
\tfrac{dR(t)}{dt} &= -2.0 \cdot r \cdot R(t) \cdot B(t) + 2.0 \cdot s \cdot T(t) \\
\tfrac{dB(t)}{dt} &= -1.0 \cdot r \cdot R(t) \cdot B(t) + 1.0 \cdot s \cdot T(t) \\
\tfrac{dT(t)}{dt} &= +1.0 \cdot r \cdot R(t) \cdot B(t) - 1.0 \cdot s \cdot T(t)
\end{aligned}$$

where $r = 0.002$ and $s = 0.2$, to be solved with respect to the initial condition $r_0 = 200$, $b_0 = 100$ and $t_0 = 500$. The numeric solution of this ODE for the above mentioned initial values is shown in Figure 4.13.

We now consider the real swarm robotics case study: the derivation of an ODE model from the Bio-PEPA specification results in a model composed of 54 ordinary differential equations.

In Figure 4.14 we can observe the total fraction of robots in the S-population over time, that is, both those present in the start area and those composing a team[6]. The fluid approximation for a model with initially 32 robots in the start

[6] To guarantee continuity of the ODE model, the H-function has been removed and replaced by setting

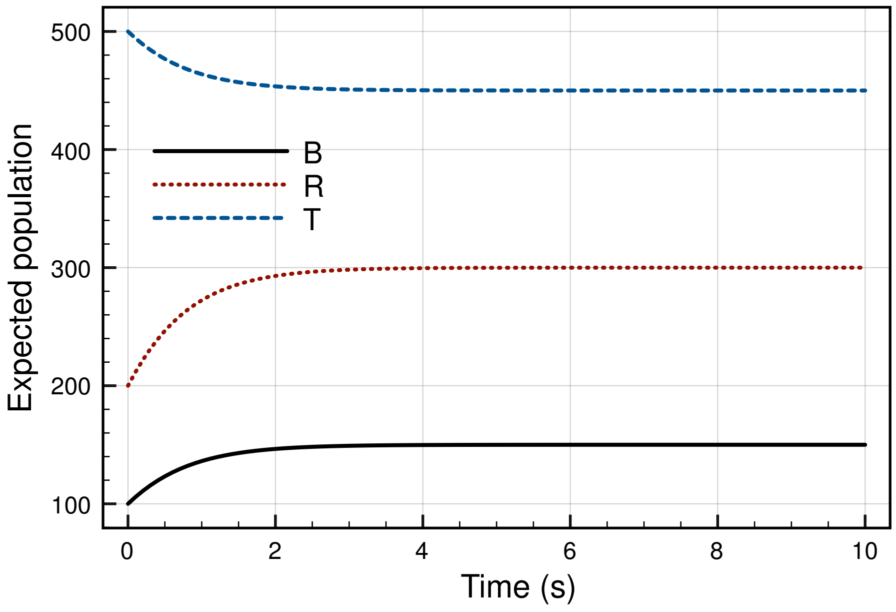


Figure 4.13: Expected population sizes of R, B and T over time (ODE) for the small toy example presented in Section 4.1.1.

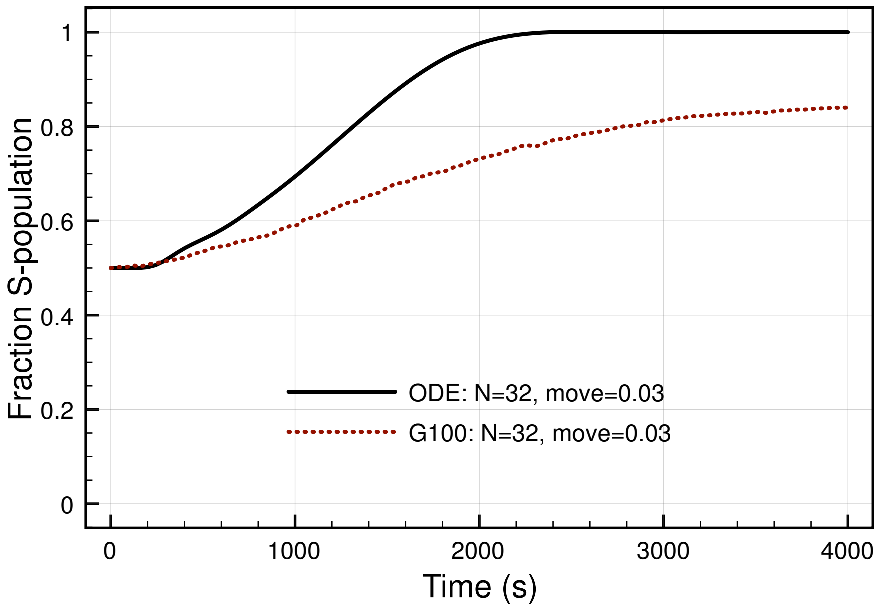


Figure 4.14: Fraction of the S-population.

move = 0.03 to approximate a scenario in which $k = 7$.

area, of which 16 would vote for the short path and 16 for the long path, predicts that the system converges in 100% of the cases to the short path for the given initial values. Stochastic simulation over 100 independent runs (G100) shows that such convergence happens only in 85% of the cases, which corresponds to what we found with statistical model checking for a comparable value of k (see Figure 4.6). The difference can be explained by the larger effect of stochastic fluctuations that occur in stochastic simulations of the system when the population is small. The probability that the system "accidentally" converges on the long path is, in that case, relatively high. In fact, if a somewhat larger population is considered, a good correspondence can be observed between the fluid approximation and stochastic simulation over 1000 independent runs (G1000), as shown in Figure 4.15 for $N = 320$.

Note that the model considered here and in the following ignores the effect of interference between robots, that is, it is assumed that the size of the paths are scaled in such a way that the density of the robots, thus the number of collisions, is kept constant and equal to the model with 32 robots for any swarm size. An ODE analysis performed on a model which considers a large number of robots can provide interesting insights in the behavior of the decision-making strategy, allowing us, at the same time, to avoid accidental stochastic fluctuations that occur with small populations.

For large populations the probability that the system "accidentally" converges to the long path tends to zero. In fact, single simulation trajectories tend to approximate the deterministic ODE solution very well for a finite time horizon when the specification satisfies certain scaling conditions and the population considered in the simulation is sufficiently large. An example is shown in Figure 4.16 for $N = 32000$. This is a well-studied phenomenon (Kurtz, 1970) which has been applied for an analysis of the double bridge experiment with ants in Bio-PEPA (Massink and Latella, 2012), for the analysis of crowd dynamics (Massink et al., 2011b) and in the context of stochastic process algebra (Tribastone et al., 2012).

A further interesting observation can be made with the help of the graph in Figure 4.16. Different phases of collective behavior can be distinguished. There is a first phase in which robots leave the start area at a constant rate. This can be observed up to ca. time 200. After that, robots begin to return to the start area, first from the short path and later on from the long path, providing feedback to the population in the start area. At about time 600, it can be observed that the feedback begins to have effect on the decision on which path to take, and an

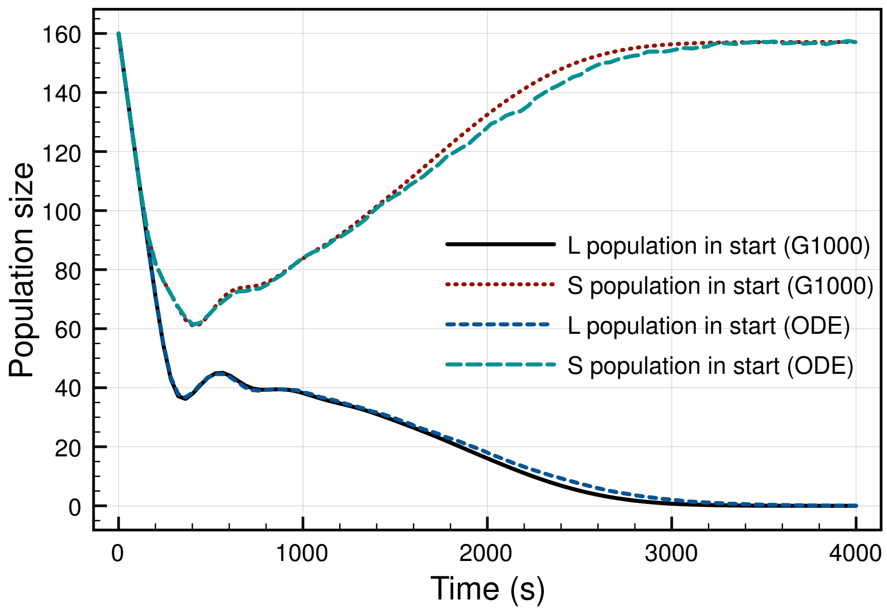


Figure 4.15: Fluid approximation (ODE) versus the mean of 1000 simulation trajectories (G1000), for $NS = NL = 160$. Parameters are $move = 0.03 * 10$ and $walk\_normal = 0.1$.

increasing number of teams take the short path rather than the long path with the consequence that the S-population in the start area continues to increase, and that of the L-population continues to decrease.

In Figure 4.17 we can observe a number of ODE trajectories for different initial values of the S-population (*NS*) and the L-population (*NL*) in the start area. The trajectories start from the points indicated on the diagonal and end in one of the two stationary points of the system indicated by a cross at (0, 15710) and at (3110, 0). Clearly, the system is bi-stable. For some initial value combination of *NS* between 12,000 and 14,000 and *NL* between 20,000 and 22,000, where the total swarm size is $NS + NL = 32,000$, a sudden shift takes place from trajectories converging on the long path to trajectories converging on the short path. In, Fig 4.17 we can also observe the effect of the last phase of the collective behavior where the feedback of the returning leads to small circle-like shapes in the curves.

Both in Figure 4.16 and Figure 4.17 the number of robots in the start area stabilizes around 15,710 in case of convergence on the short path, and on about 3,110 in case of convergence on the long path. That means that, in the former case, about 50% of the total population resides in the start area and that, on

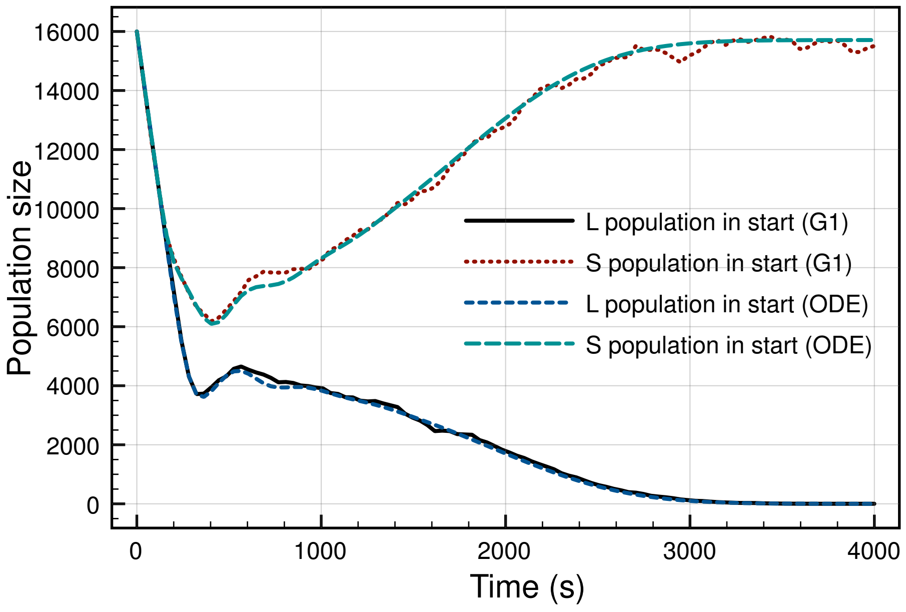


Figure 4.16: Fluid approximation (ODE) versus single simulation trajectory (G1), for $NS = NL = 16000$. Parameters are $move = 0.03 * 1000$ and $walk\_normal = 0.1$.

average, 5,430 teams circulate on the short path. In the latter case, there are far fewer robots in the start area and on average 9,630 teams circulate on the long path.

Note that Figure 4.17 has been obtained via an automatic translation of the Bio-PEPA specification into SBML (Bornstein et al., 2004), which is a standard markup language widely used in systems biology, and then via another translator[7] from SBML into the Octave (Eaton, 2002) or equivalently into the Matlab language (Gilat, 2004). Such tool-chain allows further numerical exploration of the generated ODEs with powerful applied mathematics tool suites.

In the last analysis we present, we consider the case in which both paths have equal length. This analysis allows us to verify that the system is able to converge to a single solution which depends exclusively on the initial number of robots in the S-population or in the L-population. In Figure 4.18, we show how the fractions of robots in the S-population are changing over time. The graph shows an ODE trajectory for an initial S-population of 480 over 1000 total robots in the nest and one for an initial S-population of 520 over 1000 total robots. As expected, for an initial S-population above 500 the population converges on the

[7]See `http://www.ebi.ac.uk/compneur-srv/sbml/converters/SBMLtoOctave.html`

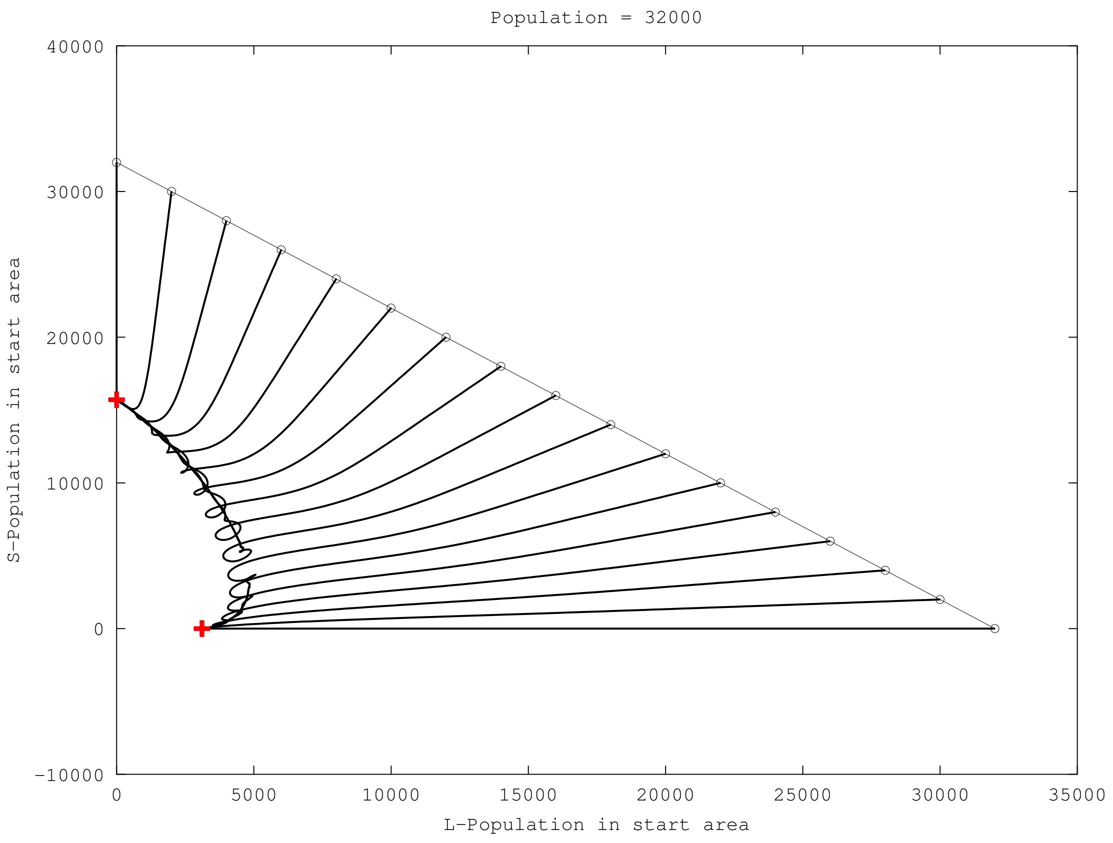


Figure 4.17: Phase-space diagram of S-population versus L-population in the start area for a population of 32,000 robots. ODE trajectories for different initial values of NS and NL starting from the diagonal line and finishing in one of the two stationary points indicated by a cross at (0, 15710) and at (3110, 0). Parameters are $move = 0.03 * 1000$ and $walk\_normal = 0.1$.

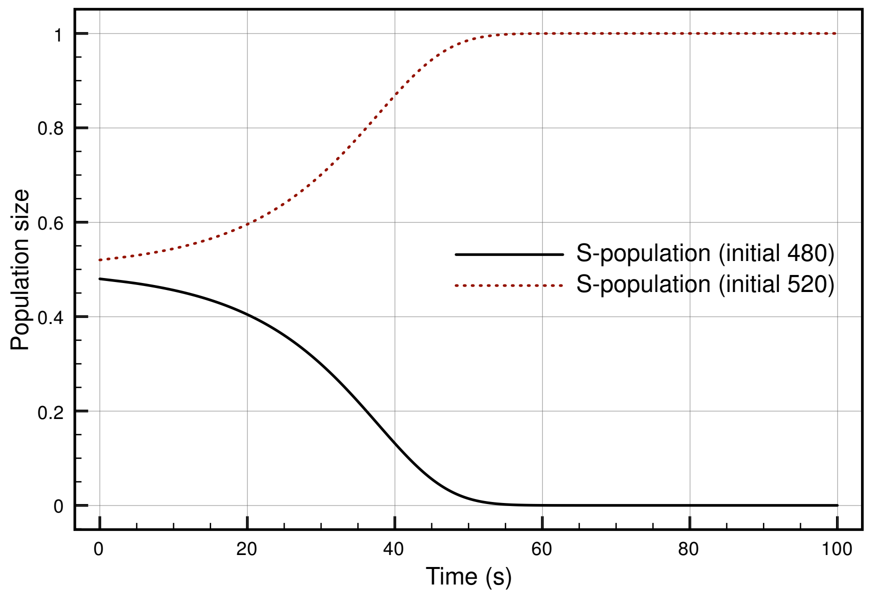


Figure 4.18: Fluid flow approximation (ODE) of the case in which path S and L have the same length and S-population is set at 480 or 520.

'short' path, and for an initial S-population below 500 it converges on the 'long' path, despite both paths are of equal length. So the population converges on the path with the highest initial consensus.

4.1.8 Summary

In this section, we analyzed a swarm robotics system using Bio-PEPA. The behavior analyzed is a decision-making behavior originally presented in Montes de Oca et al. (2011). Bio-PEPA (Ciocchetta and Hillston, 2009) is a language based on the process algebra PEPA. It was originally developed for the stochastic modeling and analysis of biochemical systems. By using Bio-PEPA we were able to model the robot swarm addressing issues like direct and indirect cooperation, team formation, heterogeneous team behaviors, voting, and certain spatial and temporal aspects.

The main advantage of the use of Bio-PEPA is that it allows the researcher to perform a variety of analyses starting from a single microscopic specification. Among the possible analyses, we performed stochastic simulation, fluid flow (ODE) approximation and statistical (stochastic) model checking. The possibility to perform different analyses from the same specification reduces the effort necessary for the analysis process, while preserving the mutual consistency of the results.

In the presented analysis, we showed that using Bio-PEPA we are able obtain results compatible with those obtained using other approaches, such as the results presented in Montes de Oca et al. (2011) via physics-based simulation and Monte Carlo simulation.

4.2 Analysis of a collective transport behavior using Klaim

In this section, we introduce a different analysis approach for robot swarms based on KLAIM and STOKLAIM.

This approach involves two phases. In the first phase, we model the behavior of the individual robots and the environment with the formal language KLAIM (De Nicola et al., 1998). KLAIM is a tuple-space-based coordination language that enables us to define an accurate model of a distributed system using a small set of primitives. In the second phase, we enrich the model with stochastic aspects, using KLAIM's stochastic extension STOKLAIM (De Nicola et al., 2007), and formalize the desired properties using MoSL (De Nicola et al., 2007). MoSL is a stochastic logic that permits specifying time-bounded probabilistic reachability properties and properties about resource distribution. The properties of interest are then verified against the STOKLAIM specifications by exploiting the analysis tool SAM (De Nicola et al., 2007, Loreti, 2013).

To demonstrate the approach, we analyze a collective transport scenario (Ferrante et al., 2013a), in which, while avoiding obstacles, a group of three robots must carry an object that is too heavy for a single robot to move. This behavior is a good candidate to establish the validity of our approach since it has many of the features that characterize collective robotic systems. Indeed, the system is completely distributed, the robots do not have any global knowledge, such as a map of the environment indicating the goal area and the position of the obstacles.

Modeling a robot swarm performing collective transport is challenging. Indeed, for understanding its dynamics, it is necessary to model in detail both the spatial aspects, that is, the positions of robots, obstacles and carried object, and the temporal aspects, that is, the robots' action execution time. Without these aspects, it would be impossible to formally verify the correctness of a collective transport behavior. Usually, the models of robot swarms presented in the literature (Lerman et al., 2005, Galstyan et al., 2005) ignore or simplify these spatial and temporal aspects in order to have smaller and simpler models that can better scale to analyze swarms composed of a large number of robots.

In order to model the collective transport behavior presented here, we sacrifice scalability for a very detailed model of the system. In collective transport this is usually not a problem, as the robots involved are usually less than 10 (see Section 2.1.3). Differently from existing approaches, which mainly focus on micro- or macroscopic aspects of the system, we employ KLAIM to capture both hardware aspects of the robots and their behavior.

KLAIM and STOKLAIM are well suited to model a robot swarm due to their compositionality and high modularity, which allow us to easily and flexibly experiment with different parameters of the scenario and of the robots' behavior. This enable us, for example, to optimize the performance of the system or prevent instabilities. Moreover, the possibility to change the parameters of the environment permits to easily check the collective behavior of the robots under different environmental conditions without the need for time-consuming experiments in simulation or with robots.

The rest of this section is structured as follows: In Section 4.2.1, we introduce the considered robotics scenario. In Section 4.2.2, we review the formal basis underlying the proposed verification approach, namely the specification language KLAIM, the stochastic extension STOKLAIM, the stochastic logic MOSL, and the analysis tool SAM. In Section 4.2.3, we describe the relevant aspects of the KLAIM specification of the scenario, while in Section 4.2.4, we present its stochastic analysis. Finally, in Section 4.2.5 we give a brief summary of the presented analysis.

4.2.1 A collective robotics scenario

In order to illustrate our approach, we consider a collective transport behavior that we originally presented in Ferrante et al. (2010a, 2013a):[8] the scenario involves three identical robots that must collectively transport an object to a goal area. See Figure 4.19 for an image of the assembled robots.

The robots operate in an arena containing obstacles. A light source indicates the position of the goal area. It is assumed that the three robots have already physically assembled to the object being carried and cannot disassemble until the goal area is reached.

The robots involved in the experiment are foot-bots. The foot-bot is an advanced mobile robot equipped with: (i) a *light sensor*, to perceive the direction to a light source; (ii) a *distance scanner*, to obtain relative distances from objects in the environment; (iii) a *range and bearing communication system*, to communicate with other robots; (iv) *wheels*, to move around the environment. More information

[8]Note that in Ferrante et al. (2013a) we presented this collective transport behavior performing only simulated experiments. A more robust and advanced version of this behavior has been developed in the context of the Swarmanoid project (Dorigo et al., 2012), where it was also tested using real robots. This upgraded version does not need a light source neither as a goal direction nor for reference. The result analyzed in this section refers to the simulated experiments presented in Ferrante et al. (2013a).

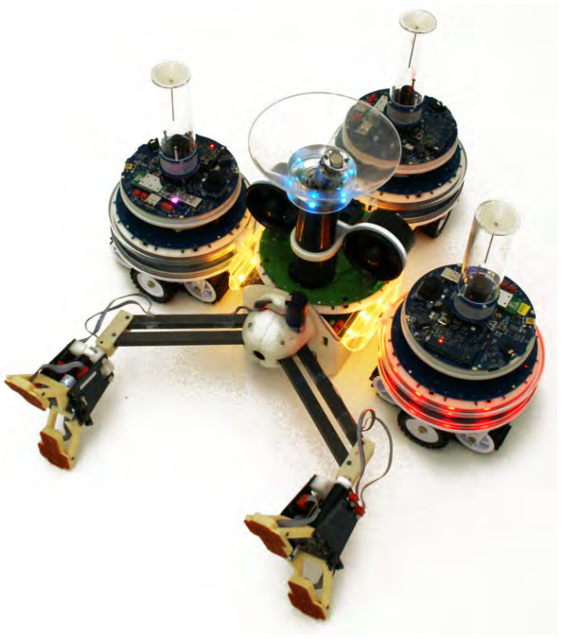
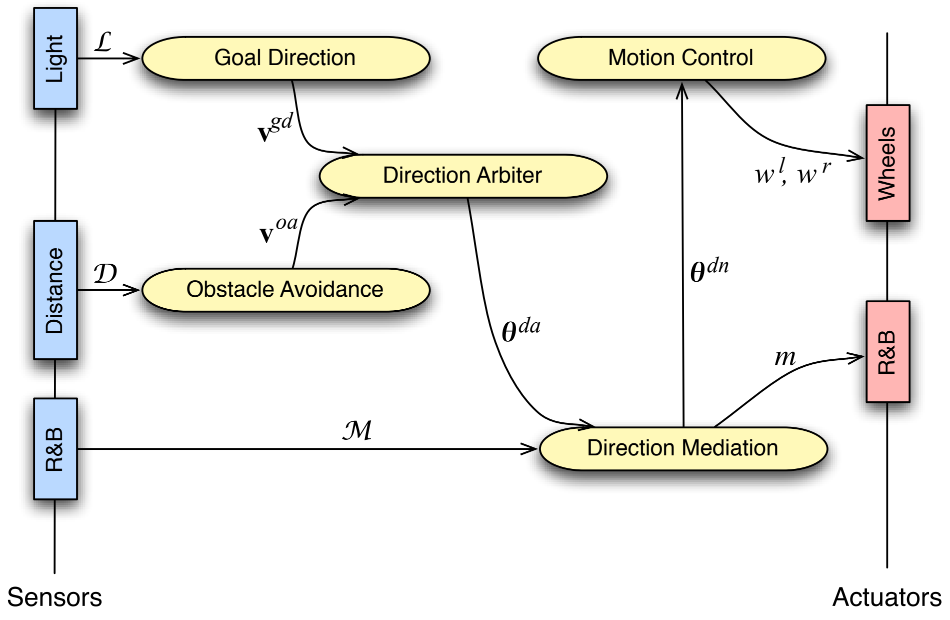


Figure 4.19: A picture that shows three foot-bots attached to the object to carry, in this case another robot of a different kind. See Dorigo et al. (2012) for more information on the use of this robot assemble.

about the foot-bot can be found in Appendix A.3.

The behavior of the individual robots is identical: each robot, based on its available information, calculates its *desired direction*, that is, the direction it would follow if it were alone. Since each robot has a local perception of the environment, the desired directions of the robots could differ. In fact, at each given moment, one robot could sense or not the position of the goal and/or the position of obstacles. According to the available information, in any given moment, a robot can be *informed*, that is, it has a desired direction to follow, or *non-informed* otherwise. Informed robots communicate to the other robots their desired direction. Finally, to actuate the wheels, all robots, informed or not, use a *socially mediated direction* obtained by averaging the received directions, so that all robots can follow the same direction even if they have a different perception of the environment. More information can be found in Ferrante et al. (2013a).

A diagrammatic description of the individual behavior, together with an explanation of the used notation, is presented in Figure 4.20. The individual behavior is composed of several modules. In the figure, each rounded rectangle is a behavioral module, whereas non-rounded rectangles are hardware modules. Each behavioral module takes an input and produces an output. The output is usually a set of variables that can be input to other modules or set as actuator



Symbol	Meaning
$\mathcal{L}$	Set of light readings
$\mathcal{D}$	Set of distances
$\mathcal{M}$	Set of received messages
m	Sent message
$\mathbf{v}^{gd}$	Vector toward to the goal area
$\mathbf{v}^{oa}$	Vector to avoid obstacles
θ^{da}	Individual desired direction
w^l/w^r	Left/right wheel speed
θ^{dn}	Mediated direction

Figure 4.20: A diagrammatic description of the individual behavior highlighting the roles of sensors and actuators and the information exchange between the different hardware and behavioral modules.

values, while the input can be the result of other modules or sensor readings. The complete individual behavior is composed of five behavioral modules, which interact with three sensors and two actuators. We now present the role of each module. Note that all angles considered in the following are relative to the position of the goal area, identified by a light source.

The *goal direction* module queries the light sensors to calculate the vector $\mathbf{v}^{gd}$. This vector originates from the center of the robot and is directed towards the position of maximum light intensity sensed, which, in this scenario, corresponds to the goal area. The length of this vector is set to 1. In case no light is perceived, the vector is set to *null*.

The *obstacle avoidance* module queries the distance scanner sensor to calculate the vector $\mathbf{v}^{oa}$. This vector originates from the center of the robot and is directed towards the points away from the closest obstacle perceived. The length of the vector $\mathbf{v}^{oa}$ corresponds to the distance to the closest object rescaled in $[0, 1]$, where 1 indicates an obstacle closer than 0.1 m and 0 indicates an obstacle at the furthest distance at which the distance scanner can perceive obstacles, that is, 1.5 m. If case no obstacle is perceived, the vector is set to *null*.

The *direction arbiter* module takes as inputs $\mathbf{v}^{gd}$ and $\mathbf{v}^{oa}$ and calculates the individual desired direction θ^{da}, that is, the desired direction of the robot before computing the mediated direction. Since the length of $\mathbf{v}^{oa}$ is proportional to how close an obstacle is, we use it to represents the urgency to avoid the obstacle. In particular, this length is used as a weight to combine the directions towards the goal area and the direction to follow to avoid obstacles into the individual desired direction. The weighted combination of $\mathbf{v}^{gd}$ and $\mathbf{v}^{oa}$ is performed only if these vectors are both non-*null*. In case both $\mathbf{v}^{gd}$ and $\mathbf{v}^{oa}$ are *null*, θ^{da} is set to *null* as well. In case only one between the two is *null*, θ^{da} is set equal to the angle of the non-*null* one.

The *direction mediation* module takes as inputs the individual desired direction θ^{da} and the received messages $\mathcal{M}$ containing the desired directions, in terms of angles, of the other robots. It produces two outputs: the first is θ^{dn}, the mediated direction. In case θ^{da} is non-*null*, the robot is said to be *informed*, and θ^{dn} is computed as the average of the individual desired direction and the desired directions of the other robots received in $\mathcal{M}$. In case θ^{da} is *null*, the robot is said to be *non-informed*, and θ^{dn} is set to the average of the desired directions received in M. The second output is m, which is the direction communicated to the other robots. If the robot is *informed*, m contains θ^{da}; if the robot is *non-informed*, m contains θ^{dn}.

The *motion control* module converts the direction θ^{dn} into the actual wheel speeds using the following equation:

$$w^l = u + \omega b \, , \, w^r = u - \omega b \, , \, \omega = K_p \theta^{dn},$$

where w^l, w^r are the rotation speed of the left/right wheel respectively, b is the distance between the center of the robot and each of the wheels, u and ω are the forward and angular velocities respectively. The forward velocity u is kept constant, whereas we vary the angular velocity ω proportionally to the socially mediated direction $\overline{\theta}_S$ to be followed. K_p is a proportional factor. For more information about motion control see Ferrante et al. (2013a).

4.2.2 Formal foundations of the verification approach

In this section, we provide a brief overview of the formal methods exploited by the proposed approach for specifying and verifying the collective transport behavior presented above.

Specification

Klaim is a process algebra whose actions are primitives taken from Linda (Carriero and Gelernter, 1989), a tuple-space model for sharing memory in distributed systems, augmented with information on the location of the nodes where the processes and tuples are allocated (De Nicola et al., 1998).

In this section, we employ a version of Klaim enriched with some standard control flow constructs (i.e., if-then-else, for and while sequence, etc.). These constructs simplify the specification task and can be easily defined in the language originally presented in De Nicola et al. (1998). Note also that, in Klaim, it is possible to define (possibly recursive) functions. All these characteristics make Klaim appear as a standard high-level programming language, which is practical for modeling software programs or generating software programs from a given Klaim specification. Nonetheless, thanks to its rigorous mathematical definition, Klaim is a complete formal specification language, which can thus be used to formally verify properties.

For simplicity, in this section we present only the operators and constructs that are used for the definition of our collective transport model. We refer to De Nicola et al. (1998) for a formal presentation of the language and to Bettini et al. (2002) for a Java framework for programming in Klaim.

In KLAIM, a distributed system is modeled as a net of *nodes*, each one with a local data repository and a set of running processes. Moreover, nodes can share data through a common data space.

Nets are finite collections of nodes composed by means of the parallel composition operator $\parallel$. Nodes are in the form $s\ ::_{\rho}\ C$, where s is a unique *locality name*, ρ is an *allocation environment*, and C is a set of components. Locality names can be used to represent, for example, a physical location or a network address. An *allocation environment* provides a name resolution mechanism by mapping *locality variables* l, which are aliases for locality names, into localities s. The distinguished locality variable **self** is used by processes to refer to the address of their current hosting node. In the rest of this section, we will use the notation ℓ to range over locality names and locality variables. *Components* are finite plain collections of tuples $\langle t \rangle$ and processes P, composed by means of the parallel operator $|$. Tuples are pieces of data shared between processes, as defined in Linda (Carriero and Gelernter, 1989).

Processes are the KLAIM active computational units and may be executed concurrently either at the same locality or at different localities. They are built up from basic actions (see below) and process calls $A(p_1, \dots, p_m)$ by means of sequential composition $P_1\,;P_2$, parallel composition $P_1 \mid P_2$, conditional choice $\textbf{if}\ (e)\ \textbf{then}\ \{P\}\ \textbf{else}\ \{Q\}$, iterative constructs $\textbf{for}\ i = n\ \textbf{to}\ m\ \{\ P\ \}$ and $\textbf{while}\ (e)\ \{P\}$, and (possibly recursive) process definitions $A(f_1, \dots, f_n) \triangleq P$ with f_i pairwise distinct. Notably, A denotes a process identifier, while f_i and p_j denote formal and actual parameters respectively, as defined below. Moreover, e ranges over *expressions*, which contain basic values (booleans, integers, strings, floats, etc.) and value variables x, and are formed by using the standard operators on basic values, simple data structures (i.e., arrays and lists) and the non-blocking retrieval actions **inp** and **readp** (explained below).

During their execution, processes perform some *basic actions*. In KLAIM, such actions are defined augmenting the actions available in the original Linda definition. Actions $\textbf{in}(T)@\ell$ and $\textbf{read}(T)@\ell$ are retrieval actions and permit to withdraw/read data tuples from the tuple space hosted at the (possibly remote) locality ℓ: if multiple matching tuples are found, one is non-deterministically chosen, otherwise the process is blocked. Actions exploit templates as patterns to select tuples in shared tuple spaces. *Tuples* t are sequences of actual fields, i.e. locality names, locality variables, expressions and processes. Instead, *templates* T are sequences of actual and formal fields, where the latter are written $!\,x$, $!\,l$ or $!\,X$ and are used to bind variables to values, locality names or processes, respectively.

Table 4.2: KLAIM operations.

KLAIM operation	Description
$\mathbf{in}(T)@\ell$	Reads and removes a tuple matching template T from location ℓ; blocking.
$\mathbf{in}(T)@\ell$	Same as above, but non-blocking.
$\mathbf{read}(T)@\ell$	Reads but does not removes a tuple matching template T from location ℓ; blocking.
$\mathbf{readp}(T)@\ell$	Same as above, but non-blocking.
$\mathbf{out}(t)@\ell$	Add tuple t to location ℓ; non-blocking
$\mathbf{eval}(P)@\ell$	Moves process P to location ℓ; non-blocking
$\mathbf{rpl}(T) \to (t)@\ell$	Replaces a tuple matching T with tuple t in location ℓ; if no tuple matches T, it behaves as $\mathbf{out}(t)@\ell$

For the sake of readability, we use "_" to denote a *wild card* formal field in a template; this corresponds to a formal field $!\,dc$ using the variable dc that does not occur elsewhere in the specification. Actions $\mathbf{inp}(T)@\ell$ and $\mathbf{readp}(T)@\ell$ are non-blocking versions of the retrieval actions: namely, during their execution processes are never blocked. Indeed, if a matching tuple is found, **inp** and **readp** act similarly to **in** and **read**, and additionally return the value *true*; otherwise they return the value *false* and the executing process does not block. $\mathbf{inp}(T)@\ell$ and $\mathbf{readp}(T)@\ell$ can be used where either a boolean expression or an action is expected (in the latter case, the returned value is simply ignored). Action $\mathbf{out}(t)@\ell$ adds the tuple resulting from the evaluation of t to the tuple space of the target node identified by ℓ, while action $\mathbf{eval}(P)@\ell$ sends the process P for execution to the (possibly remote) node identified by ℓ. Both **out** and **eval** are non-blocking actions. Action $\mathbf{rpl}(T) \to (t)@\ell$ atomically replaces a non-deterministically chosen tuple in ℓ matching the template T by the tuple t; if no tuple in ℓ matches T, the action behaves as $\mathbf{out}(t)@\ell$. Finally, action $x := e$ assigns the value of e to x and, differently from all the other actions, it is not indexed with an address because it always acts locally.

Analysis

Quantitative analysis of a KLAIM specification can be enabled by associating a rate to each action, thus obtaining a STOKLAIM (De Nicola et al., 2007) specification. This rate is the parameter of an exponentially distributed random variable accounting for the action duration time. A real valued random variable X has a *negative exponential distribution* with *rate* $\lambda > 0$ if and only if the *probability* that $X \leq t$, with $t > 0$, is $1 - e^{-\lambda \cdot t}$. The expected value of X is λ^{-1}, while its variance is λ^{-2}. The operational semantics of StoKlaim permits associating to each specification a continuous time Markov chain (CTMC) that can be used to perform quantitative analyses of the considered system.

The desired properties of a system under verification are formalised using the stochastic logic MoSL (De Nicola et al., 2007). MoSL formulae use predicates on the tuples located in the considered Klaim net to express the reachability of the system goal, or more generally, of a certain system state, while passing or not through other specific intermediate states. Therefore, MoSL can be used to express quantitative properties of the overall system behavior, such as, whether the robots are able to reach the goal or whether collisions between the robots and the obstacles ever happen in the system. The results of the evaluation of such properties do not have a rigid meaning, like *true* or *false*, but have a probabilistic nature as, for example, *in 99.7% of the cases, the robots reach the goal within t time units*.

Verification of MoSL formulae over StoKlaim specifications is assisted by the analysis tool Sam (De Nicola et al., 2007, Loreti, 2013), which uses statistical model checking (Calzolai and Loreti, 2010) to estimate the probability of the property satisfaction. In this way, the probability associated to a path-formula is determined after a set of independent observations and the algorithm guarantees that the difference between the computed value and the exact one exceeds a given *tolerance* ε with a probability that is less than a given *error probability* p.

4.2.3 Specification of the robotics scenario

In this section, we present the Klaim specification of the robots' behavior informally introduced in Section 4.2.1. We also specify the details of the robots and the arena where the robots move. We use Klaim to model also these aspects because, on the one hand, the language is expressive enough to suitably represent them and, on the other hand, this approach enables the use of existing tools for the analysis of Klaim specifications.

Here, we focus only on the *qualitative* aspects of the scenario. In the next section, our specification will be enriched with *quantitative* aspects by associating a rate to each Klaim action, thus obtaining a StoKlaim specification.

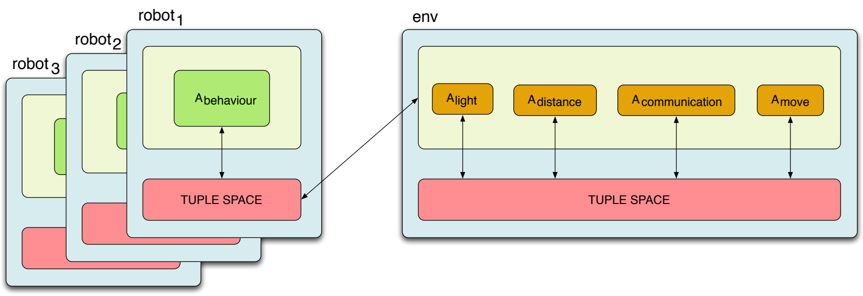


Figure 4.21: Graphical representation of the KLAIM specification

The scenario model

The overall scenario is modeled in KLAIM by the following net:

$$
\begin{aligned}
& robot_1 ::_{\{\mathbf{self} \mapsto robot_1\}} \; A_{behaviour} \mid C_{robotData\,1} \\
\parallel \; & robot_2 ::_{\{\mathbf{self} \mapsto robot_2\}} \; A_{behaviour} \mid C_{robotData\,2} \\
\parallel \; & robot_3 ::_{\{\mathbf{self} \mapsto robot_3\}} \; A_{behaviour} \mid C_{robotData\,3} \\
\parallel \; & env ::_{\{\mathbf{self} \mapsto env, r_1 \mapsto robot_1, r_2 \mapsto robot_2, r_3 \mapsto robot_3\}} \; A_{light} \mid A_{distance} \mid A_{communication} \mid A_{move} \mid C_{envData}
\end{aligned}
$$

which is graphically depicted in Figure 4.21. The three robots are modeled as three KLAIM nodes whose locality names are $robot_1$, $robot_2$ and $robot_3$. Similarly, the environment around the robots is modeled as a node, with locality name env. The allocation environment of each robot node contains only the binding for **self** (i.e., $\mathbf{self} \mapsto robot_i$), while the allocation environment of the env node contains the binding for **self** (i.e., $\mathbf{self} \mapsto env$) and the bindings for the robot nodes (i.e., $r_i \mapsto robot_i$, with $i \in \{1, 2, 3\}$).

The behavior of the individual robots is modeled as a process identified by $A_{behaviour}$, which is the same in all three robots. The items of *local knowledge* data $C_{robotData\,i}$ of each robot, that is, sensor readings and computed data, are stored in the tuple space of the corresponding node.

The processes running on the env node provide environmental data to the robots' sensors and keep this information up-to-date in time, according to the actions performed by the robots' actuators. The process A_{light}, given the position of the light source and the current position of the robots, periodically computes the information about the light position perceived by each robot and sends it to them. This data corresponds to the values obtained from light sensors and is stored in the tuple space of each robot. Similarly, the process $A_{distance}$ provides each robot with information about the obstacles around it. The process $A_{communication}$ models the communication infrastructure and, hence, takes care of delivering the messages sent by the robots by means of their range and bearing communication systems. Finally, the process A_{move} periodically updates the robots' positions according to their directions.

The data within the *env* node can be *static*, such as the information about the obstacles and the source of light, or *dynamic*, such as the robots' positions. The tuples $C_{envData}$ are stored in the tuple space of this node and their meaning is as follows: $\langle \text{“}pos\text{”}, x_1, y_1, x_2, y_2, x_3, y_3 \rangle$ represents the positions (x_1, y_1), (x_2, y_2) and (x_3, y_3) of the three robots; $\langle \text{“}light\text{”}, x_l, y_l, i \rangle$ represents a light source, with intensity i and origin in (x_l, y_l); $\langle \text{“}obstacles\text{”}, m \rangle$ indicates the total number of obstacles present in the environment; and $\langle \text{“}obs\text{”}, n, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4 \rangle$ represents the n-th rectangular-shaped obstacle, with vertices (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, y_4).

It is worth noting that, while the KLAIM process $A_{behaviour}$ is intended to model the actual robot's behavior, the KLAIM processes and data representing the robots' running environment (i.e., sensors, actuators, obstacles, goal, etc.) are just models of the environment and of physical devices needed for the analysis. In other words, the model of the environment is not meant to be realistic or descriptive, only functional to the analysis.

The model

Each robot executes an individual behavior that interacts with the robot's tuple space for reading and producing sensors and actuators data to cyclically perform the following activities: *sensing* data about the local environment, *elaborating* the retrieved data to make decisions, and *acting* according to the elaborated decisions, that is, transmitting data to other robots and actuating the wheels to move.

As introduced in Chapter 2, models of robot swarms can be developed at different scales. The majority of the models are macroscopic, that is, they consider the swarm as a whole, ignoring the individual robots. In case of collective transport, ignoring the individual robots would prevent us from understanding the interactions between the robots, which in turn, would prevent us from understanding the behavior of the swarm. Moreover, since collective transport is a collective behavior focused on the spatial interactions between the robots and the carried object, we strongly believe that it is necessary to develop a model that describes carefully both spatial and temporal aspects of the system. For this reason, we chose to develop a model at the microscopic level. Moreover, we not only consider the individual behavior of the robots, but also its constituent modules and the robots' sensors and actuators.

In this section, we illustrate the data associated to the robots' sensors and actuators, and the KLAIM specification of the individual behavior of the robots. The full specification can be found in Gjondrekaj et al. (2011).

Robots' sensor and actuator data. The *light sensor* data is modeled in KLAIM as a tuple of the form $\langle$"*light*"$, \vec{\ell}\rangle$, where "*light*" is a *tag* indicating the sensor originating the data while $\vec{\ell}$ is an array of 24 elements. For each $i \in [0, 23]$, $\vec{\ell}[i]$ represents the light intensity perceived by the sensor along the direction $2\pi\frac{i}{24}$. Process A_{light}, running in the environment node, generates a tuple containing the light sensor data for each robot.

The tuple containing the measures of the *distance scanner sensor* is similar: it is of the form $\langle$"*obs*"$, \vec{d}\rangle$, where "*obs*" is the tag associated to *distance scanner sensor* data and $\mathbf{d}$ is an array of 24 elements. For each $i \in [0, 23]$, $\vec{d}[i]$ is the distance to the closest obstacle measured by the sensor along the direction $2\pi\frac{i}{24}$. Process $A_{distance}$, running in the environment node, generates a tuple containing the distance scanner data for each robot.

The *range and bearing communication system* acts as both a sensor and an actuator: it allows a robot to send messages to other robots in its neighborhood and to receive messages sent by them. Process $A_{communication}$, running in the environment node, routes the messages produced by each robot to the other robots. This process models the communication medium and specifies the range and bearing communication system without considering explicitly all the details of the underlying communication framework. Each robot stores received messages in a local tuple of the form $\langle$"*msgs*"$, [m_1, m_2, \ldots, m_n]\rangle$ representing a queue of length n containing messages m_1,m_2,...,m_n . Instead, to send a message to the other robots, a message is locally stored as a tuple of the form $\langle$"*msg*"$, m\rangle$. The process running on the environment node is in charge of reading each message and propagating it to the other robots that are in the sender's communication range.

Finally, the *wheel actuators* are modeled as a process running in the *env* node that reads the new directions to be followed by the robots (i.e., tuples of the form $\langle$"*move*"$, \theta\rangle$) and updates the robots' position (which is, in fact, an information stored in the tuple space of the environment node). This slightly differs from the original specification given in Section 4.2.1, where the *motion control* module converts the direction calculated by the *direction mediation* module into speeds for the two wheels. This simplification does not affect the quality of the model and eases the analysis process.

Robot's behaviour. We model the individual behavior of the robots following the schema presented in Section 4.2.1 and in Figure 4.20. In particular, we define one process for each module as follows:

$$A_{behaviour} \triangleq A_{goalDirection} \mid A_{obstacleAvoidance} \mid A_{directionArbiter} \mid A_{directionMediation} \mid A_{motionControl}$$

We provide a description of each process by presenting its Klaim model. To ease comprehension the model is presented with comments, starting with // . Note that all processes are defined as recursive. This is just an artifice to represent the fact that the processes are executed continuously, as on the real robots, and do not end once completed.

The *goal direction* module is modeled as a process that takes as input the last light sensors readings and returns the vector $\mathbf{v}^{gd}$:

```
$A_{goalDirection} \triangleq$
$x_{sum}, y_{sum} := 0;$
$\mathbf{read}(\text{"}light\text{"}, !\ell)@\mathbf{self};$   // Read the tuple containing the light sensor readings
$\mathbf{for}\ i = 0\ \mathbf{to}\ 23\{$
   $x_{sum} := x_{sum} + \ell[i] \cdot \cos(2\pi i/24);$   // Calculate the coordinates of the final point of the
   $y_{sum} := y_{sum} + \ell[i] \cdot \sin(2\pi i/24);$   // vector (with the origin as initial point) resulting
$\};$                                                   // from the vectorial sum of the reading vectors

$\mathbf{if}\ ((x_{sum} != 0) \wedge (y_{sum} != 0))\ \mathbf{then}\ \{$   // Check if the light is perceived
   $\angle\mathbf{v}^{gd} := Angle(0, 0, x_{sum}, y_{sum});$   // Calculate $\angle\mathbf{v}^{gd}$, i.e., the direction of vector $\mathbf{v}^{gd}$
   $\mathbf{rpl}(\text{"}vgd\text{"}, \_) \rightarrow (\text{"}vgd\text{"}, \angle\mathbf{v}^{gd})@\mathbf{self};$   // Update the vector $\mathbf{v}^{gd}$ data
$\}\ \mathbf{else}\ \{$
   $\mathbf{inp}(\text{"}vgd\text{"}, \_)@\mathbf{self}$   // If the light is not preceived, remove the previous vector $\mathbf{v}^{gd}$ data
$\};\ A_{goalDirection}$
```

The **read** action never blocks the execution of process $A_{goalDirection}$ as sensor readings are continuously generated by the A_{light} process.

The function $Angle(x_0, y_0, x_1, y_1)$, used above and in subsequent parts of the specification, returns the direction of the vector from (x_0, y_0) to (x_1, y_1). We refer the interested reader to Loreti (2013) for its formal definition in Klaim.

The *obstacle avoidance* module is modeled as a process that takes as input the last distance sensors readings and returns the vector $\mathbf{v}^{oa}$:

```
$A_{obstacleAvoidance} \triangleq$
$x_{sum}\,,\, y_{sum} := 0\,;\;\; min := obs\_d_{MAX};$
$\mathbf{read}(\text{“}obs\text{”}, !d)@\mathbf{self}\,;$   // Read the tuple representing the distance sensor readings
$\mathbf{for}\; i = 0 \;\mathbf{to}\; 23\{$
  $x_{sum} := x_{sum} + d[i] \cdot \cos(2\pi i/24)\,;$   // Calculate the coordinates of the final point of the
  $y_{sum} := y_{sum} + d[i] \cdot \sin(2\pi i/24)\,;$   // vectorial sum of the reading vectors
  $\mathbf{if}\; (d[i] < min)\; \mathbf{then}\; min := d[i]$   // Calculate the minimum length of the vectors
$\};$
$\|\, \mathbf{v}^{oa} \,\| := min/obs\_d_{MAX}\,;$   // Calculate $\| \mathbf{v}^{oa} \|$, i.e., the length of vector $\mathbf{v}^{oa}$ rescaled in $[0,1]$
$\angle\, \mathbf{v}^{oa} := Angle(0, 0, x_{sum}, y_{sum})\,;$   // Calculate $\angle\, \mathbf{v}^{oa}$, i.e., the direction of vector $\mathbf{v}^{oa}$
$\mathbf{rpl}(\text{“}voa\text{”}, \_, \_) \rightarrow (\text{“}voa\text{”}, \| \mathbf{v}^{oa} \|, \angle\, \mathbf{v}^{oa})@\mathbf{self}\,;$   // Update the vector $\mathbf{v}^{oa}$ data
$A_{obstacleAvoidance}$
```

where obs_d_{MAX} is the maximum range of the distance sensor (in Ferrante et al. (2013a), it is set to 1.5 m).

The *direction arbiter* module is modeled as a process that takes $\mathbf{v}^{gd}$ and $\mathbf{v}^{oa}$ as input and returns the direction θ^{da}:

```
$A_{directionArbiter} \triangleq$
$\mathbf{in}(\text{“}voa\text{”}, !voa\_l, !\theta^{oa})@\mathbf{self}\,;$   // Read and consume the tuple containing $\mathbf{v}^{oa}$ (always present)
$\mathbf{if}\; (\mathbf{inp}(\text{“}vgd\text{”}, !\theta^{gd})@\mathbf{self})\; \mathbf{then}\; \{$   // Read and consume the tuple containing $\mathbf{v}^{gd}$ (if available)
  $v^{da}_x := (1 - voa\_l) \cdot cos(\theta^{oa}) + voa\_l \cdot cos(\theta^{gd})\,;$   // Calculate the coordinates of the
  $v^{da}_y := (1 - voa\_l) \cdot sin(\theta^{oa}) + voa\_l \cdot sin(\theta^{gd})\,;$   // vector to the desired direction
  $\theta^{da} := Angle(\,0,\, 0,\, v^{da}_x,\, v^{da}_y\,)\,;$   // Compute the angle $\theta^{da}$
  $\mathbf{rpl}(\text{“}da\text{”}, \_) \rightarrow (\text{“}da\text{”}, \theta^{da})@\mathbf{self}\,;$   // Update the angle $\theta^{da}$ data
$\}\; \mathbf{else}\; \{$
  $\mathbf{if}\; (voa\_l < 1)\; \mathbf{then}\; \{$   // Check if any obstacle has been detected
    $\mathbf{rpl}(\text{“}da\text{”}, \_) \rightarrow (\text{“}da\text{”}, \theta^{oa})@\mathbf{self}$   // Use the obstacle avoidance direction as $\theta^{da}$
  $\}$
$\}\,;\; A_{directionArbiter}$
```

Differently from sensor readings, data produced by other modules (e.g. $\mathbf{v}^{gd}$ and $\mathbf{v}^{oa}$) are removed from the tuple space when read.

The *direction mediation* module is modeled as a process that takes as input the direction θ^{da} computed by the process $A_{directionArbiter}$ and the last received messages from other robots and returns the direction θ^{dn}, to be used by the process

$A_{motionControl}$, and a message m, to be sent to the other robots via the $A_{communication}$ process that models the range and bearing system:

```
$A_{directionMediation} \triangleq$
$c, sum_x, sum_y := 0;$
$\mathbf{rpl}(\text{"}msgs\text{"}, !l) \rightarrow (\text{"}msgs\text{"}, [])\mathbf{@self};$  // Read and reset the list of received messages
$\mathbf{while}\ (l == \theta :: tail)\ \{$  // Scan the list
    $l := tail;$
    $sum_x := sum_x + cos(\theta);$  // Calculate the sum of the received directions
    $sum_y := sum_y + sin(\theta);$
    $c := c + 1$  // Increase the counter of the received messages
$\};$
$\mathbf{if}\ (c == 0)\ \mathbf{then}\ \{$  // If there are no received messages,
    $\mathbf{if}\ (\mathbf{inp}(\text{"}da\text{"}, !\theta^{da})\mathbf{@self})\ \mathbf{then}\ \{$  // but the robot is *informed*
        $\mathbf{rpl}(\text{"}dir\text{"}, \_) \rightarrow (\text{"}dir\text{"}, \theta^{da})\mathbf{@self};$  // Update the direction data for the motion control
        $\mathbf{rpl}(\text{"}msg\text{"}, \_) \rightarrow (\text{"}msg\text{"}, \theta^{da})\mathbf{@self}$  // Update the message to be sent to the other robots
    $\}$
$\}\ \mathbf{else}\ \{$  // If there are received messages,
    $\theta^{dn} := Angle(0, 0, sum_x, sum_y);$  // calculate the average direction and proceed
    $(\ \mathbf{rpl}(\text{"}dir\text{"}, \_) \rightarrow (\text{"}dir\text{"}, \theta^{dn})\mathbf{@self}$  // Update the data for the motion control
      $|$
      $(\ \mathbf{if}\ (\mathbf{inp}(\text{"}da\text{"}, !\theta^{da})\mathbf{@self})\ \mathbf{then}\ \{$  // If the robot is *informed*
            $m := \theta^{da}$  // send $\theta^{da}$
       $\}\ \mathbf{else}\ \{$  // If the robot is *non-informed*
            $m := \theta^{dn}$  // send $\theta^{dn}$
       $\};$
       $\mathbf{rpl}(\text{"}msg\text{"}, \_) \rightarrow (\text{"}msg\text{"}, m)\mathbf{@self}$  // Update the message to be sent to the other robots
      $)$
    $)$
$\};\ A_{directionMediation}$
```

Notice that the tuple containing the direction θ^{da} is consumed when read. Thus, to avoid blocking the execution of the process, to read such tuple an action **inp** (within the condition of an **if** construct) is exploited.

The last module is the *motion control*, which is modeled as a process that takes as input the direction computed by the process $A_{directionMediation}$ and output a tuple that is used by the process A_{move} to move the robot:

$A_{motionControl} \triangleq$
$\mathbf{in}(\text{“}dir\text{”}, !\theta^{dn})@\mathbf{self}\,;$ // wait (and consume) a direction of movement
$\mathbf{rpl}(\text{“}move\text{”}, \_) \rightarrow (\text{“}move\text{”}, \theta^{dn})@\mathbf{self}\,;$ // transmit the direction to the wheels actuator
$A_{motionControl}$

As previously explained (page 131), we do not model the conversion of the direction calculated by the *direction mediation* module into speeds for the wheels. We instead direct model its displacement.

4.2.4 Stochastic specification and analysis

In this section, we first augment the Klaim specification presented in the previous section with action rates, in order to obtain a StoKlaim stochastic model. We then use this model to perform model checking and verify properties of the collective transport behavior presented. Using statistical model checking, we first analyze the system success in moving the object in the scenario presented in Ferrante et al. (2013a), obtaining accurate estimations of the system performance expressed in terms of the probability of reaching the goal. We then change the experimental setup to analyze the performance of the collective transport behavior in different scenarios, with different obstacle and light source positions.

Modeling the system at a very detailed level allows us to analyze also what happens in case the order of execution of the modules composing the individual behavior of the robot is not deterministic. In other words, in our model, modules of different robots are not executed synchronously following a predefined order, but have follow a stochastic schedule dictated by the rate of the actions associated to each module. This is different from computer simulators, such as the one used in Ferrante et al. (2013a), where all robots act synchronously, but also from macroscopic models, in which stochasticity is used only in the action completed by the different robots. This low-level synchronization might hide possible problems caused by sensing or communication delays which could arise in real-robot experiments.

We now enrich the Klaim specification introduced in the previous section with stochastic aspects and consider the scenario presented in Ferrante et al. (2013a) and depicted in Figure 4.22. Seven rectangular objects are scattered in the arena, while the light source is positioned high above the goal area and is always visible to the robots. Following the experimental setup used in Ferrante

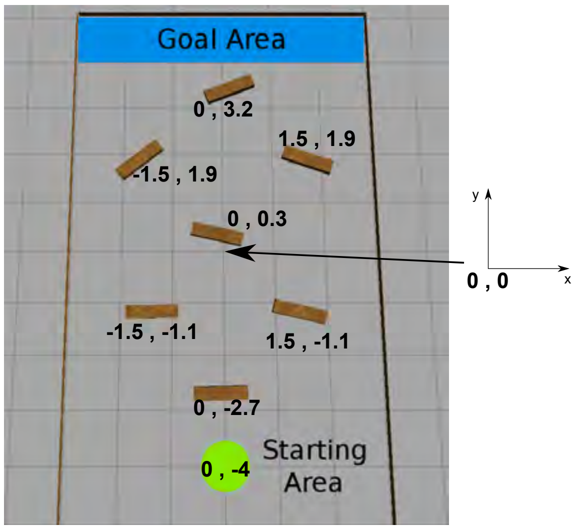


Figure 4.22: Arena and initial configuration

et al. (2013a), we assume that robots are able to perform 10 sensor readings per second on average and that they have an average speed of 0.02 m/s , and model the part of the specification modeling the environment as able to perform a mean of 100 operations per second. Starting from these parameters, we derive specific rates for defining the StoKlaim specification.

Augmenting the Klaim specification with rates to obtain a StoKlaim specification consists in annotating each tuple-manipulating operation, that is, those presented in Table 4.2 with appropriate rates. As an example, we report below the stochastic definition of process $A_{obstacleAvoidance}$, which extends the one presented in Section 4.2.3:

$$
\begin{array}{l}
A_{obstacleAvoidance} \triangleq \\
x_{sum}\,,\, y_{sum} \;:=\; 0\,;\;\; min \;:=\; obs\_d_{MAX}; \\
\textbf{read}(\text{“}obs\text{”}, !d)\textbf{@self} : \lambda_1\,; \\
\textbf{for } i = 0 \textbf{ to } 23\{\;\ldots\;\}; \\
\quad \parallel \mathbf{v}^{oa} \parallel := \; min/obs\_d_{MAX}\,;\;\; \angle\mathbf{v}^{oa} \;:=\; Angle(0, 0, x_{sum}, y_{sum}); \\
\textbf{rpl}(\text{“}voa\text{”}, \_, \_) \rightarrow (\text{“}voa\text{”}, \parallel \mathbf{v}^{oa} \parallel, \angle\mathbf{v}^{oa})\textbf{@self} : \lambda_2\,; \\
A_{obstacleAvoidance}
\end{array}
$$

The actions highlighted by a gray background are those annotated with rates λ, where $\lambda_1 = 24.0$ and $\lambda_2 = 90.0$. These rates guarantee that obstacle avoidance data are updated every $\frac{1}{24} + \frac{1}{90}$ time units on average, i.e. about 20 times per second. We refer the interested reader to Gjondrekaj et al. (2011) for the rest of the stochastic specification.

The result of a stochastic simulation run of the StoKlaim specification, performed using Sam, is reported in Figure 4.23a. The trajectories followed by the three robots in this run are plotted in the figure with three different colors. The figure shows that the robots reach the goal without collisions. On an Apple iMac computer (2.33 GHz Intel Core 2 Duo and 2 GB of memory) the simulation of a single run needs an average time of 123 seconds.

We now analyze the probability to reach the goal without colliding with any obstacles. The property *"robots have reached the goal area"* is formalized in MoSL, for the specific system under analysis, by the formula ϕ_{goal} defined below:

$$\phi_{goal} = \langle \text{"}pos\text{"}, !x_1, !y_1, !x_2, !y_2, !x_3, !y_3 \rangle @env \rightarrow y_1 \geq 4.0 \wedge y_2 \geq 4.0 \wedge y_3 \geq 4.0$$

This formula relies on the *consumption* operator, $\langle T \rangle @l \rightarrow \phi$, that is satisfied whenever a tuple matching template T is located at l and the remaining part of the system satisfies ϕ. Hence, formula ϕ_{goal} is satisfied if and only if tuple $\langle \text{"}pos\text{"}, x_1, y_1, x_2, y_2, x_3, y_3 \rangle$, where each y_i is greater than 4.0, is in the tuple space located at env (all robots are in the goal area). Similarly, the property *"a robot collided an obstacle"* is formalized by:

$$\phi_{col} = \langle \text{"}collision\text{"} \rangle @env \rightarrow \mathsf{true}$$

where tuple $\langle \text{"}collision\text{"} \rangle$ is a tuple that can be generated by the process A_{move} located at env whenever a robot collided an obstacle. See Gjondrekaj et al. (2011) for the implementation details.

The considered analyses have been then performed by estimating the total probability of the set of runs satisfying $\neg\phi_{col} U^{\leq t} \phi_{goal}$ where the formula $\phi_1 U^{\leq t} \phi_2$ is satisfied by all the runs that reach within t time units a state satisfying ϕ_2 while only traversing states that satisfy ϕ_1. In the analysis, a time-out of 500 s has been considered.

Under the experimental conditions presented in Ferrante et al. (2013a) we get that the goal can be reached without collisions with probability 0.916, while robots do not reach the goal or collide with obstacles with probability 0.084.[9] These results are in accordance with those reported in Ferrante et al. (2013a), where the estimated probability to reach the goal is 0.94. We conjecture that the slight difference in the results is mainly due to a different way of modeling robots movement, which is computed via a *physics-based computer simulator* in Ferrante

[9]These and the following values have been estimated performing statistical model checking using SAM with parameters $p = 0.1$ and $\varepsilon = 0.1$.

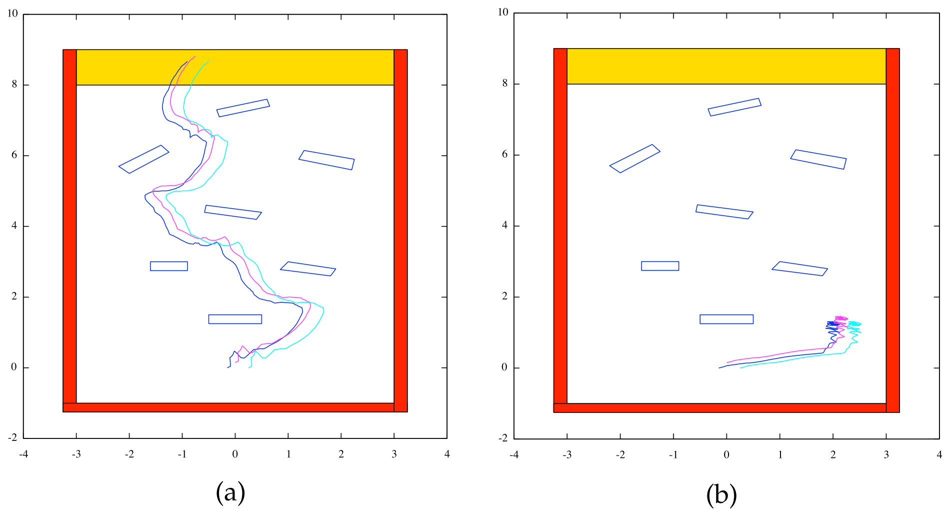


Figure 4.23: Some simulation results obtained for the robotics scenario from Ferrante et al. (2013a)

et al. (2013a), whereas in our case it is approximated as the vectorial sum of the movement of each single robot.

The next analysis we present has the goal of understanding the effect of the light source position on the behavior of the robots. To this end, we modify the original scenario by placing the light source at ground level, so that it can be obscured by objects and other robots and prevent the robots from perceiving it in some cases. The obtained results show that the behavior of the robots is drastically influenced. Under this configuration, the robots are not able to reach the goal area and the probability to reach the goal without collisions becomes close to 0.0. See Figure 4.23b for a stochastic simulation trace of a failed run.

To understand if this problem manifests itself also in other environments, we analyze the collective behavior in a environment with two obstacles and the light in the same low-height position as in the previous analysis. The environment is shown in Figure 4.24a. At the beginning, the obstacles do not hide the light to the robots. However, when the first obstacle enters in the range of the robots' distance sensors, the robots turn to right, enter in the shadow cast by the second object and then never reach the goal area.

The problem created by not having an always available light source can be avoided by modifying the individual behavior of the robots so that, when the light is not perceived, the last known goal direction is used. This approach was indeed used in the collective transport behavior developed for the Swarmanoid project (Dorigo et al., 2012). An analysis of this improved behavior shows that this

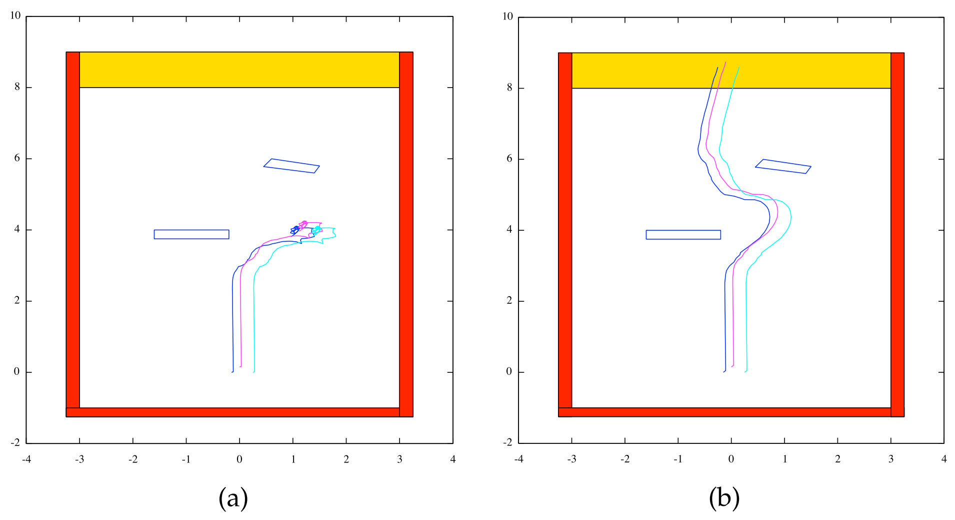


Figure 4.24: Some simulation results obtained for a simple robotics scenario

simple policy increases the probability to reach the goal area without collisions, from 0.234 to almost 1.0. A successful stochastic simulation run is presented in Figure 4.24b.

4.2.5 Summary

We have presented a novel approach to the formal verification of collective robotic systems and validated it against the traditional approach based on physics-based simulations. We first developed a Klaim model of the robot swarm. This was completed modeling the constituent modules of the individual behavior of the robots, together with their sensors and actuators. We then extended this model with rates to obtain a StoKlaim model. This model was then used to perform statistical model checking. The results of the analysis showed results which are in accordance with those produced via physics-based simulations and reported in Ferrante et al. (2013a), that have been in fact exploited for tuning the quantitative aspects of our analysis.

In this section, we presented how it is possible to develop a detailed and accurate model of a system by describing its modules using Klaim. This detailed model allowed us to formally analyze the performance of the studied collective behavior and to identify its limits. This is due to the fact that our model is able to take into account the exact positions of the robots at any given time, in addition to their sensors and actuators.

Such kind of detailed model is very well suited for collective transport scenarios, where usually a limited number of robots are involved; however, it may become intractable using available model checkers when the number of robots grows significantly. To deal with such kind of scenarios, the abstraction level of the model has to be increased in accordance with the number of robots, by focusing on those aspects of the system that become most relevant.

4.3 Discussion

In this chapter, we presented two examples of the use of model checking for the analysis of robot swarms.

In Section 4.1, we applied model checking to a collective decision-making behavior. We developed the model using Bio-PEPA, which allowed us to define a single high-level specification of the system and subsequently perform several kinds of analyses: stochastic simulation, fluid flow analysis and statistical model checking. The analysis of the system performed using model checking allowed us to formally verify various properties of the system, such as its convergence to the best path under different initial conditions.

In Section 4.2, we applied model checking to a collective transport behavior. We used Klaim, which allowed us to model in details the system, to study the robot-to-robot interactions as well as the sensor-to-actuator interactions of each single robot. The analysis performed using model checking allowed us to formally verify various properties of the system, such as its probability to successfully complete the transport of the object in different environments.

The analyses performed in this chapter show us that model checking is a viable and profitable way to analyze a robot swarms. Model checking allow us to analyze a system both at the macroscopic, as demonstrated in the collective decision-making case study, and at the microscopic level, as demonstrated in the collective transport case study.

It is worth noting that the models developed for model checking are, in general, qualitatively similar to those developed for other analysis methods as, for example, rate equations. This means that such models suffer from the same limits and drawbacks, especially considering spatial and temporal aspects of robot swarms. However, this also means that it is possible to exploit the results obtained from a large corpus of techniques and results available in the literature, see Section 2.1.

Model checking has many advantages over the use of other analysis approaches: it allow us to perform a complete analysis of a system, contrary to stochastic and physics-based simulations which are only limited to the specific instances observed. It also allow us to observe the system far from its average behavior or with a limited number of robots, something that is not possible using rate equations and fluid-flow analysis. In particular, an analysis involving the sensors and actuators of robots, as presented in Section 4.2, would be impossible using these techniques.

Concluding, in this chapter, we demonstrated that model checking can be successfully employed for the analysis of robot swarms.

Chapter 5

Conclusions

In this chapter we summarize the main contributions of this dissertation and we present future research directions concerning the design and analysis of robot swarms. In particular, we focus on the use of formal methods in swarm robotics.

In this dissertation, we showed how model checking can be successfully employed for the design and analysis of robot swarms.

In Chapter 3, we showed how model checking can be used as a tool to assist the design of a robot swarms in a novel design method called *property-driven design*. Property-driven design is a design method based on prescriptive modeling and model checking. It consists of four phases: in the first phase, the requirements of the system to design are specified in the form of properties; in the second phase, the prescriptive model of the system to design is created and improved until the desired properties are satisfied; in the third phase, the prescriptive model is used as a blueprint to implement the desired system in a physics-based simulation; in the fourth phase, the physic-based simulation is used to implement the system on real robots.

We demonstrated how property-driven design can be used to specify the requirements of a robot swarms and to realize swarms which satisfy the desired properties "by design". Property-driven design provides several advantages compared to a "code-and-fix" design approach: it allows the designer to avoid the need of "reinventing the wheel", as different models can be reused as starting points for the development of multiple robot swarms; moreover, it allows the designer to focus on the important aspects of the systems, postponing the need to deal with implementation details; finally, since most of the design iterations are done on the model, it allows the designer to minimize the work on the implementation on simulated or real robots, shortening the total time necessary to complete the system.

One of the limits of property-driven design is that, even though it provides the designer with a blueprint to implement the desired robot swarm, it is still necessary for the designer to use its ingenuity and expertise to derive the behavior of the individual robots from the model of the collective behavior. This problem could be solved, as explained in next section, with the integration of property-driven design with automatic design methods.

In Chapter 4, we showed how model checking can be used as a tool to analyze robot swarms. In particular, we used two process algebras, specifically Bio-PEPA and Klaim, to model two collective behaviors: a collective decision-making behavior and a collective transport behavior. The developed models where then analyzed using model checking.

We demonstrated how model checking can be used both on macroscopic and microscopic models. Moreover, we showed how model checking can be used to perform a complete analysis of a system, as well as to analyze the behavior of the system far from its equilibrium, thus proving a more complete analysis approach compared to stochastic and physics-based simulations and fluid-flow analysis.

Model checking is a powerful analysis approach. However, it must be noted that the quality of the performed analysis strongly depends on the quality of the model used. Modeling a robot swarm is not simple. Capturing the robot-to-robot and robot-to-environment interactions that are at the core of a robot swarm is not always easy, since it is not easy to model in detail such interactions and their temporal and spatial characteristics. Fortunately, it is possible to exploit the available techniques for modeling robot swarm available in the literature (see Chapter 2). Also in this case, the difficulty of developing models of robot swarms for model checking can be reduced using automatic technique, as explained in the next section.

Concluding, in this dissertation we demonstrated how formal methods, and in particular, model checking, can be successfully employed for the design and analysis of robot swarms.

Future work

We believe that formal methods will eventually become a standard asset in the toolbox of the swarm engineer and will help in promoting the use of swarm robotics for real-world applications. Here, we highlight some ideas that can promote the use of model checking in swarm robotics.

Guidelines for formal methods in swarm robotics

In this dissertation, we explored the use of formal methods for the design and analysis of robot swarms. Various approaches and formalisms have been presented and used, each one with distinguishing strengths and limits. As a future research, it would be useful to provide guidelines on how to choose the best suited formalism or approach for each class of collective behaviors, following the classification of collective behaviors presented in Section 2.1.3. This would promote the use of formal methods as an analysis and development tool in swarm robotics.

Automatic property-driven design

One currently adopted solution to the problem of deriving individual behaviors that result in the desired collective behavior is automatic design (see Section 2.1). One possible way to overcome the limits of property-driven design is to enhance it using automatic design: once a model of the system to realize has been completed, it can be used to guide the optimization process used to produce the behavior of the individual robots. A possible way to implement this would be to unify property-driven design and AutoMoDe (Francesca et al., 2014), an automatic design method that creates collective behaviors in the form of probabilistic finite state machines.

Automatic modeling of robot swarms using model checking

Developing a model of a robot swarms is a complex and time consuming process. A possible way to reduce the effort necessary is to use techniques such as Bayesian estimation and model comparison (Strelioff et al., 2007) to automatically identify the parameters of a predefined model or the best model out of set of given models. Through the creation of a set of predefined publicly available models, for instance a set of models for each collective behavior identified in Section 2.1, it would be possible to automatically develop a model of a system with limited intervention of the designer.

Direct verification of robot swarms

As said, the quality of the results obtainable through model checking strongly depends on the quality of the model used. One of the problems of models is that they are, by definition, a simplified view of the system being analyzed, meaning that a property valid on the model might not be valid on the modeled system, for instance because of a bug in the implementation. A way to side-step this problem would be to perform model checking directly on software without using a model. Techniques for performing model checking on high-level programming languages are already available (Visser et al., 2003). These techniques cannot be directly applied to swarm robotics, as the collective behavior of the swarm is not encoded directly, but results from the robot-to-robot and robot-to-environment interactions. However, it could be possible to formally verify the software of the individual robots and "link" it to a user-defined model of the collective behavior using statistical model checking to verify the consistency of the models.

Debugging in robot swarms through runtime verification

Debugging and faulty prevention in swarm robotics are challenging, as they suffer from the same problems related to design and analysis: the inability to predict the collective behavior from the individual behaviors. A possible solution to this problem could be to use model checking to perform runtime verification (Bauer et al., 2006): First, a model of the system under analysis is created; then, during the execution of the system, using Bayesian estimation (Epifani et al., 2009), the parameters of the model are automatic tuned to match the observation of the current execution of the system; finally, model checking is performed at runtime, and if the results obtained predict that the system will enter a faulty state, the swarm can be stopped and its behavior improved before critical faults manifest.

Online adaptation of robot swarms based on runtime verification

Runtime verification also allows us to provide the robot swarm with online adaptation. The parameter of a predefined model of the system are updated at runtime using the local or global observation of the execution of the system via Bayesian estimation (Epifani et al., 2009). With this up-to-date model it is then possible to perform runtime model checking and verify that the performance of the robots are within the expected parameters. In case this is not true, it is possible to let the swarm adapt its behavior, using the result of model checking as a feedback to guide this adaptation.

Appendix A

The robots and the simulation platform

In this appendix, we describe the tools utilized to carry out the experiments described in Chapter 3 and Chapter 4.

A.1 The e-puck robot

In this section, we introduce the e-puck robot, which has been used to perform the experiments presented in Chapter 3.

The e-puck[1] (Figure A.1) is a small wheeled robot designed for research and education (Mondada et al., 2009). It has a diameter of 7.5 cm, a height of 6 cm and a weight of 660 g. Despite its small size, the e-puck is rich in sensors and actuators.

- 8 infrared proximity sensors placed around the body of the e-puck for measuring the proximity of obstacles neighboring the e-puck.
- A 3-axis accelerometer measuring the acceleration of the e-puck and its inclination.
- 3 microphones allowing to localize the source of a sound by triangulation.
- A camera (640x480 pixels) in front of the e-puck.
- 2 stepper motors, one for each wheel, having a full rotation of 1000 steps per wheel.
- A speaker.

[1]http://www.e-puck.org/

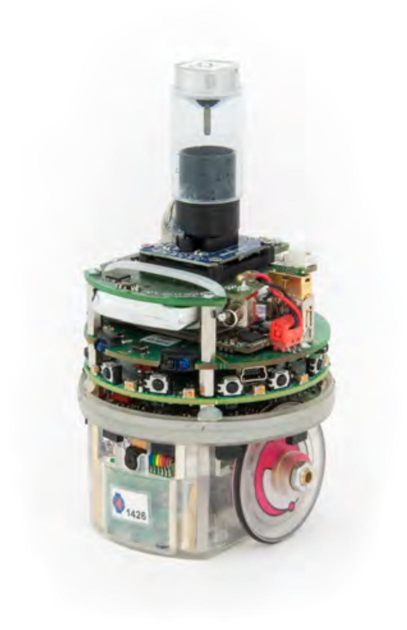


Figure A.1: The e-puck robot.

- 8 red LEDs placed around the body of the e-puck.
- One green LEDs placed in the transparent body.

One of the most interesting characteristics of the e-puck is its modularity. The e-puck can be extended using additional modules. For the experiments presented in this dissertation, we used the following additional modules:

- A ground sensor composed of three infrared sensors able to distinguish the color of the ground under the robot.
- The Overo Gumstick module,[2] a single-board computer that extends the computing capabilities of an e-puck and allows it to run linux. The Overo Gumstick also includes: 3 additional long-range infrared sensors for obstacle avoidance, 8 additional RGB LEDs and a wireless ethernet adapter.
- The Ominivision module,[3] which allows the robot to have an omnidirectional vision of its surrounding. It also include an extra battery for extended duration.
- The range and bearing module (Gutiérrez et al., 2009), which allows the

[2]http://www.gctronic.com/doc/index.php/Overo_Extension
[3]http://www.gctronic.com/doc/index.php/Omnivision_Module_V2

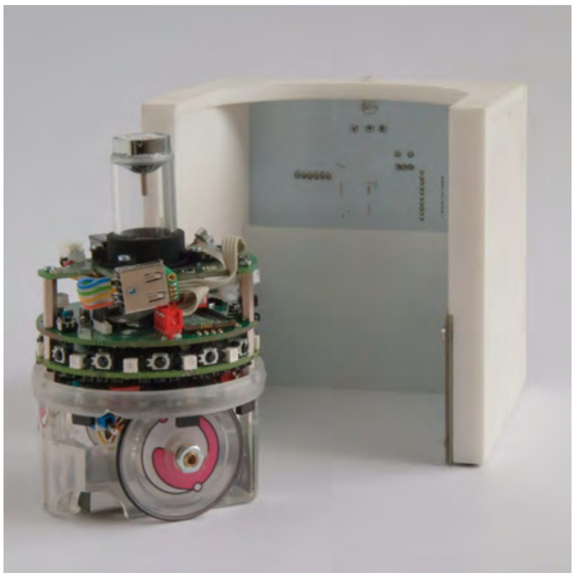


Figure A.2: The TAM together with an e-puck.

e-puck to perceive the range and bearing of neighboring robots and communicate with them.

A.2 The TAM

The TAM is essentially a booth into which an e-puck can enter (see Figure A.2 for an image). We designed the TAM to simulate the execution of a task by an e-puck even if the e-puck does not have the necessary capabilities to complete the task. For example, it can be used to simulate the deactivation of a land mine by an e-puck even if an e-puck is clearly not equipped for this task. The simulation is performed simply by letting the e-puck wait for a specific time inside a TAM.

Physically, the TAM has a cubical shape with a length of 12 cm in every dimension. We designed the weight of the body of the TAM so that an e-puck can enter into the TAM without accidentally moving it. The e-puck can perceive the LEDs of the TAM only from an acute angle: Experiments have shown that the TAM can be recognized by the e-pucks from a distance up to 80 cm and an angle of $45 \pm 4°$.

The TAM is based on *Arduino*[4], an open-source experimental platform that uses an Atmel AVR micro-controller as central processor. It is controlled by a central 8-bit RISC processor, an ATmega328P running at 16 MHz, which is

[4]http://www.arduino.cc/

equipped with 2kB main memory and 32kB flash memory. The central processor controls the sensors and actuators of the TAM.

The TAM has two IR light barriers, an IR transceiver and three RGB LEDs. The IR light barriers are used to detect the presence of a robot in the TAM. The IR transceiver of the TAM is used to communicate with a robot that has entered the TAM by using the e-puck library *IRcom*[5].

The communication between the TAM and the central computer is wireless, implemented using a 2.4 GHz XBee mesh networking module. Mesh networking enables experiments with a large number of TAMs allowing scalability.

A.3 The foot-bot robot

In this section, we introduce the foot-bot robot, which has been used to perform the experiments presented in Chapter 4.

The foot-bot robot is a modular robot composed of many sensors and actuators, collectively referred to as modules. Each module is controlled by a dedicated dsPIC micro-controller. The foot-bot is equipped with a main processor board, a Freescale i.MX31 ARM 11 low-energy 533 MHz processor running linux. The main board features 128 MB of DDR RAM and 64 MB of flash. The dsPICs on the modules communicate with the central processor asynchronously using a common bus and the ASEBA software platform (Magnenat et al., 2011).

Figure A.3 shows a picture of the foot-bot where only the sensors and the actuators used in this dissertation are marked. The foot-bot is 29 cm tall and has a radius of 8.5 cm. Its weight is 1.8 Kg. It uses a lithium polymer battery with very long duration and that can be hot-swapped during an experiment thanks to the presence of a super capacitor.

The complete list of robot's sensor and actuators is the following:

- Two differential drive *treels*, a combination of tracks and wheels, are used for locomotion in normal and rough terrains.

- A turret actuator allows a plastic ring with 12 RGB LEDs and a gripper to rotate 360 degrees around the vertical axis of the robot. The ring is used both to emit light with different colors and as a docking mechanism for another foot-bots. In fact, the gripper is especially designed to fit this specific plastic ring and to hold into it while open.

[5] http://gna.org/projects/e-puck/

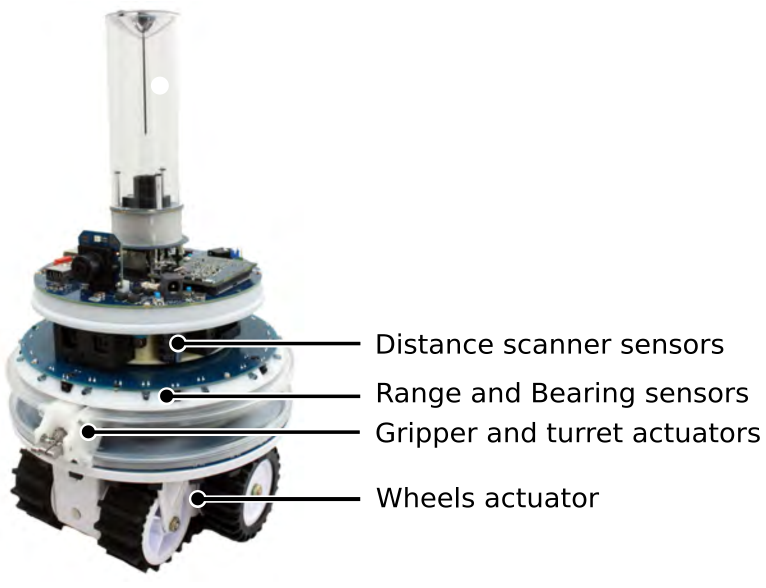


Figure A.3: The foot-bot robot. In evidence the sensors and actuators used in the experiment presented in this dissertation.

- 24 infrared sensors, that are evenly distributed around the foot-bot's body, have two functionalities. First, as a proximity sensor, they can detect obstacles in close range (5 cm). Second, as a light sensor, they can measure the intensity and the direction of the ambient light, even when placed far away from the robot.

- 4+8 additional infrared sensors, usually referred to as ground sensors, are located underneath the robot, between the two tracks and on the outside part, respectively. They are used to detect the color of the ground in the gray scale.

- Two Pixelplus 2.0 MegaPixels CMOS cameras provide basic visual information. The first camera is located in the above part of the robot, in the center. It point upwards towards a mirror located at the top of a glass tube. This provides the foot-bot with omni-directional vision capabilities. The second camera is also located on the top but in a non-central position, and can be installed to either look upwards towards the ceiling or forward along the *X*-*Y* plane.

- A rotating scanner, composed of two short and two long distance scanners

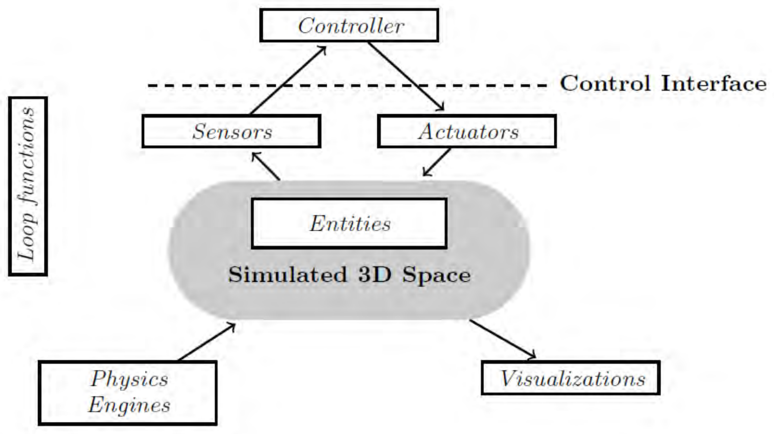


Figure A.4: Architecture of the ARGoS simulator.

based on infrared sensors, is used to measure distances. It can detect distance to obstacles very precisely, at the expenses of a poor 360 degrees resolution due to the limited rotation speed.

- A range and bearing sensing and communication board. It is composed of 20 infrared transmitters, of 12 receivers, and of a radio communication device. The infrared transmitters and receivers have two functionalities. First, they are used to detect the range and the bearing of neighboring foot-bots. Second, they are used for local communication. This module is only used for communication in this dissertation.
- A three-axes accelerometer and a three-axes gyroscope installed on the left treel.

A.4 The ARGoS simulator

All the experiments described this dissertation have been carried out also using a physics-based simulator called ARGoS, which stands for *Autonomous Robots Go Swarming* (Pinciroli et al., 2011, 2012). Argos is an open source simulator[6] specifically designed for research in swarm robotics. It was developed in the context of the Swarmanoid project.

The overall architecture of ARGoS is shown in Figure A.4. ARGoS has been developed with two key concepts in mind: *flexibility* and *efficiency*. Flexibility refers to the possibility to tune the experiments according to the needs. Efficiency

[6]ARGoS simulator, http://iridia.ulb.ac.be/argos, February 2013.

refers to the possibility to perform experiments with large numbers of robots and achive high performance in term of execution speed. Flexibility has been achieved through a modular design of the simulator: all components, such as robots, sensors, actuators, physics engines, visualization engines, are plugins that can be freely selected and included/excluded. In addition, efficiency in ARGoS has been pursued through parallelization, that is, multiple sensors or multiple physics engine can run in parallel on multiple cores. Parallelization of the physics engine has been achieved via partitioning of the simulated space into non-overlapping sub-spaces and assignment of each sub-space to a separate physics engine.

One of the key features of ARGoS, it is that the control software written for testing on ARGoS can be seamlessly instantiated on real robots without the need for any kind of modification or translation. This is achieved via a control interface (Figure A.4), which is an abstraction layer between the controller and the sensors/actuators. The user can then decide whether to compile the control software for the simulation or for the real robot.

Bibliography

Abbott, R. (2006). Emergence explained. *Complexity*, 12(1):13–26. 2.1.2

Agassounon, W. and Martinoli, A. (2002). Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1090–1097. IFAAMAS, Richland, SC. 2.1.3

Amé, J., Halloy, J., Rivault, C., Detrain, C., and Deneubourg, J. L. (2006). Collegial decision making based on social amplification leads to optimal group formation. *Proceedings of the National Academy of Sciences*, 103(15):5835–5840. 2.1.3, 2.1.3

Ampatzis, C. (2008). *On the Evolution of Autonomous Time-based Decision-making and Communication in Collective Robotics*. PhD thesis, IRIDIA, Université Libre de Bruxelles, Belgium. 2.1.1

Ampatzis, C., Tuci, E., Trianni, V., Christensen, A. L., and Dorigo, M. (2009). Evolving self-assembly in autonomous homogeneous robots: Experiments with two physical robots. *Artificial Life*, 15:465–484. 2.1.3

Ampatzis, C., Tuci, E., Trianni, V., and Dorigo, M. (2008). Evolution of signaling in a multi-robot system: Categorization and communication. *Adaptive Behavior*, 16(1):5–26. 2.1.1

Anderson, C., Theraulaz, G., and Deneubourg, J.-L. (2002). Self-assemblages in insect societies. *Insectes Sociaux*, 49(2):99–110. 2.1.3

Bachrach, J., Beal, J., and McLurkin, J. (2010). Composable continuous-space programs for robotic swarms. *Neural Computation and Applications*, 19(6):825–847. 2.1.1

Bahçeci, E. and Şahin, E. (2005). Evolving aggregation behaviors for swarm robotic systems: a systematic case study. In *Proceedings of the 2005 Swarm Intelligence Symposium (SIS)*, pages 333–340. IEEE Press, Piscataway, NJ. 2.1.3

Bahçeci, E., Soysal, O., and Şahin, E. (2003). A review: pattern formation and adaptation in multi-robot systems. Technical Report CMU-RI-TR-03-43, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. 2.1.3

Balch, T. and Hybinette, M. (2000). Social potentials for scalable multi-robot formations. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA)*, pages 73–80. IEEE Press, Piscataway, NJ. 2.1.3

Baldassarre, G., Nolfi, S., and Parisi, D. (2003). Evolving mobile robots able to display collective behaviors. *Artificial Life*, 9(3):255–267. 2.1.3

Baldassarre, G., Parisi, D., and Nolfi, S. (2006). Distributed coordination of simulated robots based on self-organization. *Artificial Life*, 12(3):289–311. 2.1.3

Baldassarre, G., Trianni, V., Bonani, M., Mondada, F., Dorigo, M., and Nolfi, S. (2007). Self-organized coordinated motion in groups of physically connected robots. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 37(1):224–239. 2.1.1

Banzhaf, W. and Pillay, N. (2007). Why complex systems engineering needs biological development. *Complexity*, 13(2):12–21. 1

Bauer, A., Leucker, M., and Schallhart, C. (2006). Model-based runtime analysis of distributed reactive systems. In *Proceedings of the Australian Software Engineering Conference (ASWEC)*, page 10. IEEE press. 5

Beal, J. (2004). Programming an amorphous computational medium. In *Proceedings of the International Workshop on Unconventional Programming Paradigms (UPP)*, volume 3566 of *Lecture Notes in Computer Science*, pages 97–97. Springer, Berlin, Heidelberg. 2.1.1

Beck, K. (2003). *Test-driven Development: By Example*. Addison-Wesley, Boston, MA. 3.2

Beckers, R., Holland, O., and Deneubourg, J.-L. (1994). From local actions to global tasks: stigmergy and collective robotics. In *Artificial life IV*, pages 181–189. MIT Press. 2.1.3

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamic neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122. 2.1.1

Beni, G. (2005). From swarm intelligence to swarm robotics. In *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 1–9. Springer, Berlin, Heidelberg. 2.1

Benkirane, S., Norman, R., Scott, E., and Shankland, C. (2012). Measles epidemics and PEPA: An exploration of historic disease dynamics using process algebra. In *Formal Methods*, volume 7436 of *Lecture Notes in Computer Science*, pages 101–115. Springer, Berlin, Heidelberg. 4.1

Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2010). *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, Berlin, Heidelberg. 2.2

Bergstra, J., Ponse, A., and Smolka, S., editors (2001). *Handbook of Process Algebra*. Elsevier Science Publishers, Amsterdam, The Netherlands. 2.2.1

Bergstra, J. A. and Klop, J. W. (1984). Process algebra for synchronous communication. *Information and Control*, 60(1–3):109–137. 2.2.1

Berman, S., Halász, Á. M., Hsieh, M. A., and Kumar, V. (2009). Optimized stochastic policies for task allocation in swarms of robots. *IEEE Transactions on Robotics*, 25(4):927–937. 2.1.1, 2.1.2

Berman, S., Kumar, V., and Nagpal, R. (2011a). Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–385. IEEE press. 2.1.1

Berman, S., Lindsey, Q., Sakar, M., Kumar, V., and Pratt, S. (2011b). Experimental study and modeling of group retrieval in ants as an approach to collective transport in swarm robotic systems. *Proceedings of the IEEE*, 99(9):1470–1481. 2.1.3

Berman, S., Nagpal, R., and Halasz, A. (2011c). Optimization of stochastic strategies for spatially inhomogeneous robot swarms: A case study in commercial pollination. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3923–3930. 2.1.2

Bettini, L., De Nicola, R., and Pugliese, R. (2002). KLAVA: A Java package for distributed and mobile applications. *Software: Practice and Experience*, 32(14):1365–1394. 4.2.2

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York. 2.1

Bordini, R. (2009). *Multi-Agent Programming: Languages, Tools and Applications*, volume 2. Springer, New York, NY. 1

Bornstein, B., Doyle, J., Finney, A., Funahashi, A., Hucka, M., Keating, S., Kovitz, H. K. B., Matthews, J., Shapiro, B., and Schilstra, M. (2004). Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (SBML) project. *Systems Biology*, 1:4153. 4.1.7

Brambilla, M., Dorigo, M., and Birattari, M. (2014a). Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking. *ACM Transactions on Autonomous and Adaptive Systems*. Submitted for publication. 3.1

Brambilla, M., Dorigo, M., and Birattari, M. (2014b). Property-driven design for robot swarms: Supplementary material. Supplementary material available at `http://iridia.ulb.ac.be/supp/IridiaSupp2014-003/index.html`. 3.3.1, 3.3.2, 3.3.2

Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41. 2.1.1, 2.2.1, 3.1, 3.3, ii, 3.3.2, 3.3.3, 4.1.2

Brambilla, M., Pinciroli, C., Birattari, M., and Dorigo, M. (2009). A reliable distributed algorithm for group size estimation with minimal communication requirements. In *Fourteenth International Conference on Advanced Robotics (ICAR)*, page 6. Proceedings on CD-ROM, paper ID 137. 2.1.3

Brambilla, M., Pinciroli, C., Birattari, M., and Dorigo, M. (2012). Property-driven design for swarm robotics. In *Proceedings of AAMAS*, pages 139–146. IFAAMAS. 3.1

Breder Jr, C. M. (1954). Equations descriptive of fish schools and other animal aggregations. *Ecology*, 35(3):361–370. 2.1.3

Brinksma, E. and Hermanns, H. (2001). Process algebra and markov chains. In *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 183–231. Springer, Berlin, Heidelberg. 2.2, 2.2.1

Brooks, R. (1990). Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15. 2.1.2

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23. 2.1.1

Bruni, R., Corradini, A., Gadducci, F., Lluch Lafuente, A., and Vandin, A. (2012). Modelling and analyzing adaptive self-assembly strategies with MAUDE. In *Rewriting logic and its applications*, volume 7571 of *Lecture Notes in Computer Science*, pages 118–138. Springer, Berlin, Heidelberg. 2.2

Brutschy, A., Pini, G., Baiboun, N., Decugnière, A., and Birattari, M. (2010). The IRIDIA-TAM: A device for task abstraction for the e-puck robot. Technical Report TR/IRIDIA/2010-015, IRIDIA, ULB. 3.3.2

Brutschy, A., Pini, G., and Decugnière, A. (2012). Grippable objects for the foot-bot. Technical Report TR/IRIDIA/2012-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. 2.1.3

Burch, J., Clarke, E., McMillan, K., and Dill, D. (1990). Sequential circuit verification using symbolic model checking. In *Proceedings of the 27th Design Automation Conference*, pages 46–51, Washington, DC. IEEE. 2.2

Calzolai, F. and Loreti, M. (2010). Simulation and analysis of distributed systems in Klaim. In *Coordination Models and Languages*, volume 6116 of *Lecture Notes in Computer Science*, pages 122–136. Springer, Berlin, Heidelberg. 4.2.2

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2001). *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, Princeton, NJ. 2.1.3, 2.1.3, 2.1.3

Campo, A. and Dorigo, M. (2007). Efficient multi-foraging in swarm robotics. In *Advances in Artificial Life, Proceedings of ECAL 2007*, volume 4648 of *Lecture Notes in Artificial Intelligence*, pages 696–705. Springer, Berlin, Heidelberg. 2.1.2

Campo, A., Garnier, S., Dédriche, O., Zekkri, M., and Dorigo, M. (2011). Self-organized discrimination of resources. *PLoS ONE*, 6(5). 2.1.3

Campo, A., Nouyan, S., Birattari, M., Groß, R., and Dorigo, M. (2006). Enhancing cooperative transport using negotiation of goal direction. In *Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS)*, volume 4150 of *Lecture Notes in Compure Science*, pages 365–366. Springer, Berlin, Heidelberg. 2.1.3

Carriero, N. and Gelernter, D. (1989). Linda in context. *Communications of the ACM*, 32(4):444–458. 4.2.2

Çelikkanat, H. and Şahin, E. (2010). Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6):849–865. 2.1.2, 2.1.3

Christensen, A. L., O'Grady, R., and Dorigo, M. (2008). SWARMORPH-script: a language for arbitrary morphology generation in self-assembling robots. *Swarm Intelligence*, 2(2–4):143–165. 2.1.3

Christensen, A. L., O'Grady, R., and Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766. 2.1.3

Ciesinski, F. and Größer, M. (2004). On probabilistic computation tree logic. In *Validation of Stochastic Systems*, volume 2925 of *Lecture Notes in Computer Science*, pages 333–355. Springer, Berlin, Heidelberg. 2.2.2

Ciocchetta, F., Duguid, A., Gilmore, S., Guerriero, M. L., and J., H. (2009). The Bio-PEPA Tool Suite. In *Proceedings of the 6th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 309–310. IEEE Computer Society, Washington, DC. 4.1.1, 4.1.6, 4.1.7

Ciocchetta, F. and Hillston, J. (2008). Bio-PEPA: An extension of the process algebra PEPA for biochemical networks. *Electronic Notes in Theoretical Computer Science*, 194(3):103–117. 4.1.1

Ciocchetta, F. and Hillston, J. (2009). Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084. 4.1, 4.1.1, 1, 4.1.1, 4.1.6, 4.1.7, 4.1.8

Ciocchetta, F. and Hillston, J. (2012). Bio-PEPA. http://www.biopepa.org, Last checked on October 2012. 4.1

Clarke, E. (1997). Model checking. In *Foundations of Software Technology and Theoretical Computer Science*, number 1346 in Lecture Notes in Computer Science, pages 54–56. Springer, Berlin, Heidelberg. 2.2, 2.2.2

Correll, N. (2008). Parameter estimation and optimal control of swarm-robotic systems: a case study in distributed task allocation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3302–3307. 2.1.2

Correll, N. and Martinoli, A. (2007). Modeling self-organized aggregation in a swarm of miniature robots. In *IEEE International Conference on Robotics and Automation*. 2.1.3

Couzin, I. D., Krause, J., Franks, N. R., and Levin, S. A. (2005). Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516. 2.1.3

Dantu, K., Berman, S., Kate, B., and Nagpal, R. (2012). A comparison of deterministic and stochastic approaches for allocating spatially dependent tasks in micro-aerial vehicle collectives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2.1.2

De Nicola, R., Ferrari, G. L., and Pugliese, R. (1998). KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330. 4.2, 4.2.2

De Nicola, R., Katoen, J.-P., Latella, D., Loreti, M., and Massink, M. (2007). Model checking mobile stochastic logic. *Theoretical Computer Science*, 382(1):42–70. 4.2, 4.2.2

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J. M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168. 2.1.3

Di Caro, G. A., Ducatelle, F., and Gambardella, L. M. (2009). Wireless communications for distributed navigation in robot swarms. In *Applications of Evolutionary Computing*, volume 5484 of *Lecture Notes in Computer Science*, pages 21–30. Springer, Berlin, Heidelberg. 2.1.3

Dixon, C., Winfield, A., and Fisher, M. (2011). Towards temporal verification of emergent behaviours in swarm robotic systems. In *Towards Autonomous Robotic Systems*, volume 6856 of *LNCS*, pages 336–347. Springer, Berlin, Heidelberg. 2.1.2, 2.2.1, 4.1

Donald, B. R., Jennings, J., and Rus, D. (1997). Information invariants for distributed manipulation. *The International Journal of Robotics Research*, 16(5):673–702. 2.1.3

Dorigo, M. and Birattari, M. (2007). Swarm intelligence. *Scholarpedia*, 2(9):1462. 2.1

Dorigo, M., Birattari, M., and Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, 9(1):1463. 1, 2.1

Dorigo, M. and Şahin, E. (2004). Guest editorial. *Autonomous Robots*, 17:111–113. 2.1

Dorigo, M., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A., Decugnière, A., Di Caro, G., Ducatelle, F., Ferrante, E., Förster, A., Martinez Gonzales, J., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Rétornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stützle, T., Trianni, V., Tuci, E., Turgut, A. E., and Vaussard, F. (2012). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4):60–71. 2.1.3, 2.1.3, 8, 4.19, 4.2.4

Dorigo, M., Tuci, E., Trianni, V., Gro, R., Nouyan, S., Ampatzis, C., Labella, T. H., O'Grady, R., Bonani, M., and Mondada, F. (2006). SWARM-BOT: Design and implementation of colonies of self-assembling robots. In *Computational Intelligence: Principles and Practice*, chapter 6, pages 103–135. IEEE press, New York, NY. 2.1.3

Ducatelle, F., Di Caro, G. A., Pinciroli, C., Mondada, F., and Gambardella, L. M. (2011a). Communication assisted navigation in robotic swarms: self-organization and cooperation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4981–4988. IEEE press, Los Alamitos, CA. 2.1.3

Ducatelle, F., G. A. Di Caro, C. P., and Gambardella, L. M. (2011b). Self-organized cooperation between robotic swarms. *Swarm Intelligence*, 5(2):73–96. 2.1.3

Eaton, J. W. (2002). *GNU Octave Manual*. Network Theory Ltd., London, UK. 4.1.7

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211. 2.1.1

Epifani, I., Ghezzi, C., Mirandola, R., and Tamburrelli, G. (2009). Model evolution by run-time parameter adaptation. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pages 111–121. IEEE press. 5

Fang, Y. (2001). Hyper-erlang distribution model and its application in wireless mobile networks. *Wireless Networks*, 7(3):211–219. 2.2.1

Ferrante, E., Brambilla, M., Birattari, M., and Dorigo, M. (2010a). Look out!: Socially-mediated obstacle avoidance in collective transport. In *Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS)*, number 6234 in Lecture Notes in Computer Science, pages 572–573. Springer, Berlin, Heidelberg. 4.2.1

Ferrante, E., Brambilla, M., Birattari, M., and Dorigo, M. (2013a). Socially-mediated negotiation for obstacle avoidance in collective transport. In *Proceedings of the International Symposium on Distributed Autonomous Robotics Systems (DARS)*, volume 83 of *Springer Tracts in Advanced Robotics Series*, pages 571–583. Springer, Berlin, Heidelberg. 2.1.3, 4.2, 4.2.1, 8, 4.2.1, 4.2.1, 4.2.3, 4.2.4, 4.2.4, 4.23, 4.2.5

Ferrante, E., Duenez-Guzman, E., Turgut, A. E., and Wenseleers, T. (2013b). GESwarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proceeding of the Genetic and evolutionary computation conference (GECCO)*, pages 17–24. ACM, New York, NY. 2.1.1

Ferrante, E., Turgut, A. E., Huepe, C., Stranieri, A., Pinciroli, C., and Dorigo, M. (2012). Self-organized flocking with a mobile robot swarm: a novel motion control method. *Adaptive Behavior*. 2.1.3

Ferrante, E., Turgut, A. E., Mathews, N., Birattari, M., and Dorigo, M. (2010b). Flocking in stationary and non-stationary environments: a novel communication strategy for heading alignment. In *Parallel Problem Solving from Nature – PPSN XI: 11th International Conference*, volume 6239 of *Lecture Notes in Computer Science*, pages 331–340. Springer, Berlin, Heidelberg. 2.1.3

Fine, T. L. (1999). *Feedforward Neural Network Methodology*. Springer, Berlin, Heidelberg. 2.1.1

Fisher, M. and Wooldridge, M. (1997). On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 06(01):37–65. 2.2

Flocchini, P., Prencipe, G., Santoro, N., and Widmayer, P. (2008). Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447. 2.1.3

Fokkink, W. (2000). *Introduction to Process Algebra*. Springer, New York, NY. 2.2.1, 2.2.1

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., and Birattari, M. (2014). AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*. In press. 2.1.1, 5

Francesca, G., Brambilla, M., Trianni, V., Dorigo, M., and Birattari, M. (2012). Analysing an evolved robotic behaviour using a biological model of collegial decision making. In *Proceedings of the 12th International Conference on Adaptive Behavior (SAB)*, volume 7426 of *Lecture Notes in Computer Science*, pages 381–390. Springer, Berlin, Heidelberg. 2.1.3

Franks, N. and Sendova-Franks, A. (1992). Brood sorting by ants: distributing the workload over the work-surface. *Behavioral Ecology and Sociobiology*, 30:109–123. 2.1.3

Friedmann, M. (2010). *Simulation of Autonomous Robot Teams with Adaptable Level of Abstraction*. PhD thesis, University of Darmstadt, Germany. 2.1.2

Frigg, R. and Hartmann, S. (2012). Models in science. In *The Stanford Encyclopedia of Philosophy*. Stanford University press, Stanford, CA, spring 2012 edition. 2.1.2

Galstyan, A., Hogg, T., and Lerman, K. (2005). Modeling and mathematical analysis of swarms of microscopic robots. In *Proceedings of the 2005 Swarm Intelligence Symposium - (SIS)*, pages 201–208. IEEE Press, Los Alamitos, CA. 2.1.2, 4.2

Garnier, S., Gautrais, J., Asadpour, M., Jost, C., and Theraulaz, G. (2009). Self-organized aggregation triggers collective decision making in a group of cockroach-like robots. *Adaptive Behavior*, 17(2):109–133. 2.1.3

Garnier, S., Jost, C., Jeanson, R., Gautrais, J., Asadpour, M., Caprari, G., and Theraulaz, G. (2005). Aggregation behaviour as a source of collective decision in a group of cockroach-like robots. In *Advances in Artificial Life*, volume 3630 of *Lecture Notes in Artificial Intelligence*, pages 169–178. Springer, Berlin, Heidelberg. 2.1.3, 2.1.3

Gazi, V. and Passino, K. M. (2002). Stability analysis of social foraging swarms: Combined effects of attractant/repellent profiles. In *Proceedings of the 41st IEEE Conference On Decision and Control*, volume 3, pages 2848–2853. IEEE Press, Piscataway, NJ. iii

Gazi, V. and Passino, K. M. (2003). Stability analysis of swarms. *IEEE Transactions on Automatic Control*, 48(4):692–696. 2.1.2

Gazi, V. and Passino, K. M. (2004a). A class of attractions/repulsion functions for stable swarm aggregations. *International Journal of Control*, 77(18):1567–1579. 2.1.2

Gazi, V. and Passino, K. M. (2004b). Stability analysis of social foraging swarms. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 34(1):539–557. 2.1.2

Gazi, V. and Passino, K. M. (2005). Stability of a one-dimensional discrete-time asynchronous swarm. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 35(4):834–841. 2.1.2

Getling, A. V. (1998). *Rayleigh-Bénard convection: structures and dynamics*, volume 11. World Scientific, London, UK. 2.1.3

Gilat, A. (2004). *MATLAB: An Introduction with Applications*. Wiley press, Hoboken, NJ. 4.1.7

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *Physical Chemistry*, 81(25):2340–2361. 4.1.1, 4.1.5

Giusti, A., Nagi, J., Gambardella, L., and Caro, G. D. (2012). Distributed consensus for interaction between humans and mobile robot swarms. In *Proceedings of 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Richland, SC. 2.1.3

Gjondrekaj, E., Loreti, M., Pugliese, R., Tiezzi, F., Pinciroli, C., Brambilla, M., Birattari, M., and Dorigo, M. (2011). Towards a formal verification methodology for collective robotic systems. Technical report, Universitá di Firenze, Florence, Italy. `http://rap.dsi.unifi.it/~loreti/papers/collective_transport_verification.pdf`. 4.2.3, 4.2.4

Gjondrekaj, E., Loreti, M., Pugliese, R., Tiezzi, F., Pinciroli, C., Brambilla, M., Birattari, M., and Dorigo, M. (2012). Towards a formal verification methodology for collective robotic systems. In *Formal Methods and Software Engineering*, volume 7635 of *Lecture Notes in Computer Science*, pages 54–70. Springer, Berlin, Heidelberg. 2.1.2, 4

Goldberg, D. and Mataric, M. J. (2001). Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In *Robot Teams: From Diversity to Polymorphism*, pages 315–344. Peters press, Natick, MA. 1

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA. 2.1.1

Goss, S., Aron, S., Deneubourg, J., and Pasteels, J. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76:579–581. 4.1.2

Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–80. 2.1.3

Groß, R. and Dorigo, M. (2008a). Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305. 2.1.1

Groß, R. and Dorigo, M. (2008b). Self-assembly at the macroscopic scale. *Proceedings of the IEEE*, 96(9):1490–1508. 2.1.3

Groß, R. and Dorigo, M. (2009). Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, 1(1–2):1–13. 2.1.3, 2.1.3

Grünbaum, D. and Okubo, A. (1994). Modeling social animal aggregations. *Frontiers in Theoretical Biology*, 100:296–325. 2.1.3

Gutiérrez, A., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., and Magdalena, L. (2009). Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3111–3116. IEEE Press, Piscataway, NJ. 3.3.1, A.1

Gutiérrez, Á., Campo, A., Monasterio-Huelin, F., Magdalena, L., and Dorigo, M. (2010). Collective decision-making based on social odometry. *Neural Computing and Applications*, 19(6):807–823. 2.1.3

Halász, A., Liang, Y., Hsieh, M., and Lai, H.-J. (2012). Emergence of specialization in a swarm of robots. In *Distributed Autonomous Robotic Systems*, volume 83 of *Springer Tracts in Advanced Robotics*, pages 403–416. Springer, Berlin, Heidelberg. 2.1.3

Hamann, H. (2012). Towards swarm calculus: Universal properties of swarm performance and collective decisions. In *Proceedings of the International Conference on Swarm Intelligence (ANTS)*, volume 7461 of *Lecture Notes in Computer Science*, pages 168–179. Springer, Berlin, Heidelberg. 2.1.2

Hamann, H. and Wörn, H. (2008). A framework of space-time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, 2(2):209–239. 2.1.1, 2.1.2

Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535. 2.2.2, 2.2.2

Havelund, K., Lowry, M., and Penix, J. (2001). Formal analysis of a space-craft controller using spin. *IEEE Transactions on Software Engineering*, 27(8):749–765. 2.2

Hillston, J. (1996). A compositional approach to performance modelling. Distinguished Dissertation in Computer Science. Cambridge University Press. Cambridge, MA. 4.1.1

Hillston, J. (2005). Fluid flow approximation of PEPA models. In *Proceedings of the 2th International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 33–43. IEEE press, Washington, DC. 4.1.1, 4.1.7

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Cambridge, MA. 2.1.1

Howard, A., Matarić, M. J., and Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 299–308. Springer, Berlin, Heidelberg. 2.1.3

Hsieh, M. A., Halász, Á., Berman, S., and Kumar, V. (2008). Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence*, 2(2–4):121–141. 2.1.2

Jeanson, R., Rivault, C., Deneubourg, J.-L., Blanco, S., Fournier, R., Jost, C., and Theraulaz, G. (2005). Self-organized aggregation in cockroaches. *Animal Behaviour*, 69(1):169–180. 2.1.3, 3.3.1

Jeyaraman, S., Tsourdos, A., Zbikowski, R., and White, B. (2005). Formal techniques for the modelling and validation of a co-operating UAV team that uses Dubins set for path planning. In *Proceedings of the American Control Conference*, volume 7, pages 4690–4695. IEEE press, Piscataway, NJ. 2.2

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134. ii

Kaminka, G. A., Schechter-Glick, R., and Sadov, V. (2008). Using sensor morphology for multirobot formations. *IEEE Transactions on Robotics*, 24(2):271–282. 2.1.3

Kazadi, S. (2000). *Swarm Engineering*. PhD thesis, California Institute of Technology, Pasadeba, CA, USA. 2.1

Kazadi, S., Lee, J. R., and Lee, J. (2009). Model independence in swarm robotics. *International Journal of Intelligent Computing and Cybernetics, Special Issue on Swarm Robotics*, 2(4):672–694. 2.1.1

Kendall, D. G. (1966). Branching processes since 1873. *Journal of London Mathematics Society*, 41(1):386–406. 2.1.2

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98. 2.1.1

Kleinrock, L. (1975). *Queueing Systems. Volume 1: Theory*. Wiley press, Hoboken, NJ. 4.1.3

Kolling, A., Nunnally, S., and Lewis, M. (2012). Towards human control of robot swarms. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 89–96. ACM, New York, NY. 2.1.3

Konur, S., Dixon, C., and Fisher, M. (2012). Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60(2):199–213. 2.1.2, 2.2

Kramer, J. and Scheutz, M. (2007). Development environments for autonomous mobile robots: a survey. *Autonomous Robots*, 22(2):101–132. 2.1.2

Krieger, M. J. B. and Billeter, J.-B. (2000). The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1–2):65–84. 2.1.3

Kube, C. R. and Bonabeau, E. (2000). Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1–2):85–101. 2.1.3

Kurtz, T. (1970). Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7:49–58. 4.1.7

Kwiatkowska, M., Norman, G., and Parker, D. (2004). Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142. 2.2.3, 3.3

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 585–591. Springer, Berlin, Heidelberg. 4.1.1

Labella, T. H., Dorigo, M., and Deneubourg, J.-L. (2006). Division of labour in a group of robots inspired by ants' foraging behaviour. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25. 2.1.1

Langer, J. S. (1980). Instabilities and pattern formation in crystal growth. *Reviews of Modern Physics*, 52(1):1–28. 2.1.3

Lerman, K. and Galstyan, A. (2002). Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots*, 13(2):127–141. 2.1.2, 2.1.3, 2.1.4, 3.3.2

Lerman, K., Galstyan, A., Martinoli, A., and Ijspeert, A. J. (2001). A macroscopic analytical model of collaboration in distributed robotic systems. *Artificial Life*, 7(4):375–393. 2.1.2, 2.1.2

Lerman, K., Martinoli, A., and Galstyan, A. (2005). A review of probabilistic macroscopic models for swarm robotic systems. In *Swarm robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 143–152. Springer, Berlin, Heidelberg. 3.3.1, 3.3.3, 4.1, 4.2

Levi, P. and Kernbach, S. (2010). *Symbiotic Multi-Robot Organisms*. Springer, Berlin, Heidelberg. 2.1.3

Li, L., Martinoli, A., and Abu-Mostafa, Y. S. (2004). Learning and measuring specialization in collaborative swarm systems. *Adaptive Behavior*, 12(3-4):199–212. 2.1.1

Lindsey, Q., Mellinger, D., and Kumar, V. (2012). Construction with quadrotor teams. *Autonomous Robots*, 33:323–336. 2.1.4

Liu, W. and Winfield, A. (2010). Modeling and optimization of adaptive foraging in swarm robotic systems. *International Journal of Robotics Research*, 29(14):1743–1760. 2.1.2

Liu, W., Winfield, A. F. T., Sa, J., Chen, J., and Dou, L. (2007). Towards energy optimization: emergent task allocation in a swarm of foraging robots. *Adaptive Behavior*, 15(3):289–305. 2.1.1, 2.1.3

Liu, Y. and Passino, K. M. (2004). Stable social foraging swarms in a noisy environment. *IEEE Transactions on Automatic Control*, 49(1):30–44. 2.1.2

Liu, Y., Passino, K. M., and Polycarpou, M. M. (2003). Stability analysis of m-dimensional asynchronous swarms with a fixed communication topology. *IEEE Transactions on Automatic Control*, 48(1):76–95. 2.1.2

Loreti, M. (2013). SAM: Stochastic analyser for mobility. 4.2, 4.2.2, 4.2.3

Magnenat, S., Rtornaz, P., Bonani, M., Longchamp, V., and Mondada, F. (2011). ASEBA: A modular architecture for event-based control of complex robots. *IEEE/ASME Transactions on Mechatronics*, 16(2):321–329. A.3

Martinoli, A., Easton, K., and Agassounon, W. (2004). Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *The International Journal of Robotics Research*, 23(4-5):415–436. 2.1.2

Martinoli, A., Ijspeert, A. J., and Mondada, F. (1999). Understanding collective aggregation mechanisms: from probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, 29(1):51–63. 2.1.2

Massink, M., Brambilla, M., Latella, D., Dorigo, M., and Birattari, M. (2012). Analysing robot swarm decision-making with Bio-PEPA. In *Swarm Intelligence*, volume 7461 of *Lecture Notes in Computer Science*, pages 25–36. Springer, Berlin, Heidelberg. 4

Massink, M., Brambilla, M., Latella, D., Dorigo, M., and Birattari, M. (2013). On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics. *Swarm Intelligence*, 7(2–3):201–228. 2.1.2, 4

Massink, M. and Latella, D. (2012). Fluid analysis of foraging ants. In *Coordination Models and Languages - 14th International Conference, COORDINATION 2012, Stockholm, Sweden, June 14-15, 2012. Proceedings*, volume 7274 of *Lecture Notes in Computer Science*, pages 152–165. Springer, Berlin, Heidelberg. 4.1.7

Massink, M., Latella, D., Bracciali, A., Harrison, M., and Hillston, J. (2011a). Scalable context-dependent analysis of emergency egress models. *Formal Aspects of Computing*, pages 1–36. In press. 4.1

Massink, M., Latella, D., Bracciali, A., and Hillston, J. (2011b). Modelling non-linear crowd dynamics in Bio-PEPA. In *Fundamental Approaches to Software Engineering*, volume 6603 of *Lecture Notes in Computer Science*, pages 96–110. Springer, Berlin, Heidelberg. 4.1, 4.1.7

Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83. 2.1.1

Matarić, M. J. (1998). Using communication to reduce locality in distributed multi-agent learning. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):357–369. 2.1.1

Mathews, N., Christensen, A. L., Ferrante, E., O'Grady, R., and Dorigo, M. (2010). Establishing spatially targeted communication in a heterogeneous robot swarm. In *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 939–946, Richland, SC. IFAAMAS. 2.1.2

Mathews, N., Christensen, A. L., O'Grady, R., and Dorigo, M. (2012). Spatially targeted communication and self-assembly. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2679, Los Alamitos, CA, USA. IEEE Computer Society Press. 2.1.3

Maxim, P. M., Spears, W. M., and Spears, D. F. (2009). Robotic chain formations. In *Proceedings of the IFAC Workshop on Networked Robotics*, pages 19–24. Elsevier, Oxford, UK. 2.1.3

McLurkin, J., Smith, J., Frankel, J., Sotkowitz, D., Blau, D., and Schmidt, B. (2006). Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In *2006 AAAI Spring Symposium*, pages 72–75. AAAI. 2.1.3

Meinhardt, H. (1982). *Models of biological pattern formation*, volume 6. Academic Press, London, UK. 2.1.3

Melhuish, C., Holland, O., and Hoddell, S. (1999a). Convoying: using chorusing for the formation of travelling groups of minimal agents. *Robotics and Autonomous Systems*, 28(2–3):207–216. 2.1.3

Melhuish, C., Welsby, J., and Edwards, C. (1999b). Using templates for defensive wall building with autonomous mobile ant-like robots. In *Proceedings of Towards Intelligent and Autonomous Mobile Robots*, volume 99. 2.1.3

Miller, J. and Mukerji, J. (2003). Mda guide version 1.0.1. `http://www.omg.org`. 3.1

Mondada, F., Bonani, M., Guignard, A., Magnenat, S., Studer, C., and Floreano, D. (2005). Superlinear physical performances in a SWARM-BOT. In *Proceedings of the VIIIth European Conference on Artificial Life (ECAL)*, volume 3630 of *Lecture Notes in Computer Science*, pages 282–291. Springer, Berlin, Heidelberg. 2.1.3

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65. IPCB. 3.3.1, 3.3.2, A.1

Montes de Oca, M. A., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., and Dorigo, M. (2011). Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. *Swarm Intelligence*, 5(3–4):305–327. 2.1.3, 4.1, 4.1.2, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6, 4.1.8

Naghsh, A., Gancet, J., Tanoto, A., and Roast, C. (2008). Analysis and design of human-robot swarm interaction in firefighting. In *Proceedings of the 17th IEEE International Symposium on the Robot and Human Interactive Communication, (Ro-Man)*, pages 255–260. 2.1.3

Nimal, V. (2010). Statistical approaches for probabilistic model checking. MSc Mini-project Dissertation, Oxford University Computing Laboratory. 2.2.3

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. Intelligent Robots and Autonomous Agents. MIT Press, Cambridge, MA. 2.1.1

Norris, J. R. (1998). *Markov Chains*. Cambridge University Press. 2.2.1

Nouyan, S., Campo, A., and Dorigo, M. (2008). Path formation in a robot swarm: Self-organized strategies to find your way home. *Swarm Intelligence*, 2(1):1–23. 2.1.1, 2.1.3

Nouyan, S., Groß, R., Bonani, M., Mondada, F., and Dorigo, M. (2009). Teamwork in self-organized robot colonies. *IEEE Transactions on Evolutionary Computation*, 13(4):695–711. 2.1.3, 2.1.3

O'Grady, R., Christensen, A., and Dorigo, M. (2009). SWARMORPH: Multi-robot morphogenesis using directional self-assembly. *IEEE Transactions on Robotics*, 25(3):738–743. 2.1.3

O'Grady, R., Groß, R., Christensen, A. L., and Dorigo, M. (2010). Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots*, 28(4):439–455. 2.1.2, 2.1.3

O'Hara, K. J. and Balch, T. (2007). *Pervasive sensor-less networks for cooperative multi-robot tasks*, pages 305–314. Distributed Autonomous Robotic Systems. Springer, Tokio, Japan. 2.1.3

Okubo, A. (1986). Dynamical aspects of animal grouping: swarms, schools, flocks, and herds. *Advances in Biophysics*, 22(0):1–94. 2.1.3

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434. 2.1.1

Parker, C. A. C. and Zhang, H. (2011). Biologically inspired collective comparisons by robotic swarms. *International Journal of Robotics Research*, 30(5):524–535. 2.1.3

Parker, L. E. (1996). L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, 11(4):305–322. 2.1.1

Parrish, J. K., Viscido, S. V., and Grünbaum, D. (2002). Self-organized fish schools: An examination of emergent properties. *Biological Bulletin*, 202(3):296–305. 2.1.3

Payton, D., Daily, M., Estowski, R., Howard, M., and Lee, C. (2001). Pheromone robotics. *Autonomous Robots*, 11(3):319–324. 2.1.2, 2.1.3

Pelánek, R. (2009). Fighting state space explosion: Review and evaluation. *Formal Methods for Industrial Critical Systems*, pages 37–52. 2.2.1, 2.2.3

Pelez, A. L. and Kyriakou, D. (2008). Robots, genes and bytes: Technology development and social changes towards the year 2020. *Technological Forecasting and Social Change*, 75(8):1176–1201. 1

Pinciroli, C., O'Grady, R., Christensen, A. L., Birattari, M., and Dorigo, M. (2013). Parallel formation of differently sized groups in a robotic swarm. *SICE Journal of the Society of Instrument and Control Engineers*, 52(3):213–226. 2.1.2, 2.1.3

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G. D., Ducatelle, F., Stirling, T., Gutierrez, A., Gambardella, L. M., and Dorigo, M. (2011). ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5027–5034. IEEE press, Los Alamitos, CA. A.4

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295. 2.1.2, 3.3.1, 3.3.2, A.4

Pini, G., Brutschy, A., Birattari, M., and Dorigo, M. (2009). Interference reduction through task partitioning in a robotic swarm. In *Sixth International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 52–59. INSTICC Press. 2.1.3, 3.3.2

Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., and Birattari, M. (2011). Task partitioning in swarms of robots: An adaptive method for strategy selection. *Swarm Intelligence*, 5(3–4):283–304. 2.1.3

Pini, G. and Tuci, E. (2008). On the design of neuro-controllers for individual and social learning behaviour in autonomous robots: An evolutionary approach. *Connection Science*, 20(2–3):211–230. 2.1.1

Podevijn, G., O'Grady, R., and Dorigo, M. (2012). Self-organised feedback in human swarm interaction. In *Proceedings of the Workshop on Robot Feedback in Human-Robot Interaction: How to Make a Robot Readable for a Human Interaction Partner (Ro-Man)*. 2.1.3

Prorok, A., Correll, N., and Martinoli, A. (2011). Multi-level spatial modeling for stochastic distributed robotic systems. *The International Journal of Robotics Research*, 30(5):574–589. 2.1.2

Pugh, J. and Martinoli, A. (2007). Parallel learning in heterogeneous multi-robot swarms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3839–3846. IEEE press. 2.1.1

Rayman, M. D., Varghese, P., Lehman, D. H., and Livesay, L. L. (2000). Results from the deep space 1 technology validation mission. *Acta Astronautica*, 47(29):475–487. 2.2

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34. 2.1.3

Sahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, Berlin, Heidelberg. 2.1

Scheidler, A. (2011). Dynamics of majority rule with differential latencies. *Phys. Rev. E*, 83:031116. 4.1, 4.1.2

Schmickl, T., Hamann, H., Wörn, H., and Crailsheim, K. (2009). Two different approaches to a macroscopic model of a bio-inspired robotic swarm. *Robotics and Autonomous Systems*, 57(9):913–921. 2.1.2

Schwager, M., Michael, N., Kumar, V., and Rus, D. (2011). Time scales and stability in networked multi-robot systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3855–3862. 2.1.2

Serfozo, R. F. (1979). An equivalence between continuous and discrete time markov decision processes. *Operations Research*, 27(3):616–620. 2.2.1

Shucker, B. and Bennett, J. K. (2007). Scalable control of distributed robotic macrosensors. In *Distributed Autonomous Robotic Systems 6*, pages 379–388. Springer, Tokyo, Japan. 2.1.3

Shucker, B., Murphey, T., and Bennett, J. (2008). Convergence-preserving switching for topology-dependent decentralized systems. *IEEE Transactions on Robotics*, 24(6):1405–1415. 2.1.3

Smith, J. M. (1978). *Models in Ecology*. Cambridge University Press, Cambridge, MA. 3.3.3

Soysal, O., Bahçeci, E., and Şahin, E. (2007). Aggregation in swarm robotic systems: Evolution and probabilistic control. *Turkish Journal of Electrical Engineering and Computer Sciences*, 15(2):199–225. 2.1.3

Soysal, O. and Şahin, E. (2005). Probabilistic aggregation strategies in swarm robotic systems. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 325–332. IEEE Press, Piscataway, NJ. 2.1.1, 2.1.3

Soysal, O. and Şahin, E. (2007). A macroscopic model for self-organized aggregation in swarm robotic systems. In *Swarm Robotics*, volume 4433 of *Lecture Notes in Computer Science*, pages 27–42. Springer, Berlin, Heidelberg. 2.1.2, 2.1.3

Spears, W. M. and Spears, D. F. (2012). *Physics-Based Swarm Intelligence*. Springer, Berlin, Heidelberg, Berlin, Heidelberg. 2.1.3

Spears, W. M., Spears, D. F., Hamann, J. C., and Heil, R. (2004). Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2–3):137–162. 2.1.1, 2.1.2, 2.1.3

Sperati, V., Trianni, V., and Nolfi, S. (2008). Evolving coordinated group behaviours through maximization of mean mutual information. *Swarm Intelligence*, 2(2–4):73–95. 2.1.1

Sperati, V., Trianni, V., and Nolfi, S. (2011). Self-organised path formation in a swarm of robots. *Swarm Intelligence*, 5:97–119. 2.1.3

Stankovic, J. A. (1996). Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, 28(4):751–763. 2.2

Stewart, R. L. and Russell, R. A. (2006). A distributed feedback mechanism to regulate wall construction by a robotic swarm. *Adaptive Behavior*, 14:21–51. 2.1.3

Stirling, T. and Floreano, D. (2010). Energy efficient swarm deployment for search in unknown environments. In *Proceedings of the 7th International Conference on Swarm Intelligence (ANTS)*, Lecture Notes in Computer Science, pages 562–563. Springer, Berlin, Heidelberg. 2.1.3

Stranieri, A., Ferrante, E., Turgut, A. E., Trianni, V., Pinciroli, C., Birattari, M., and Dorigo, M. (2011). Self-organized flocking with a heterogeneous mobile robot swarm. In *Advances in Artificial Life, ECAL 2011*, pages 789–796. MIT press, Cambridge, MA. 2.1.3

Strelioff, C. C., Crutchfield, J. P., and Hübler, A. W. (2007). Inferring Markov chains: Bayesian estimation, model comparison, entropy rate, and out-of-class modeling. *Physical Review E*, 76(1). 5

Theraulaz, G., Bonabeau, E., and Deneubourg, J.-L. (1998). Response threshold reinforcements and division of labour in insect societies. *Proceedings of the Royal Society B: Biological Sciences*, 265(1393):327–332. 2.1.3

Trianni, V. and Dorigo, M. (2006). Self-organisation and communication in groups of simulated and physical robots. *Biological Cybernetics*, 95:213–231. 2.1.1

Trianni, V., Groß, R., Labella, T. H., Şahin, E., and Dorigo, M. (2003). Evolving aggregation behaviors in a swarm of robots. In *Advances in Artificial Life: 7th European Conference – (ECAL)*, volume 2801 of *Lecture Notes in Artificial Intelligence*, pages 865–874. Springer, Berlin, Heidelberg. 2.1.3

Trianni, V., Labella, T. H., Groß, R., Şahin, E., Dorigo, M., and Deneubourg, J.-L. (2002). Modeling pattern formation in a swarm of self-assembling robots. Technical Report TR/IRIDIA/2002-12, IRIDIA, Université Libre de Bruxelles, Belgium. 2.1.2

Tribastone, M., Gilmore, S., and Hillston, J. (2012). Scalable differential analysis of process algebra models. *IEEE Trans. Software Eng.*, 38(1):205–219. 4.1.7

Tschaikowski, M. and Tribastone, M. (2012). Exact fluid lumpability for Markovian process algebra. In *CONCUR 2012 - Concurrency Theory. 23rd International Conference*, volume 7454 of *Lecture Notes in Computer Science*, pages 380–394. Springer, Berlin, Heidelberg. 4.1.1

Tuci, E., Trianni, V., and Dorigo, M. (2004). 'Feeling' the flow of time through sensorymotor coordination. *Connection Science*, 16(4):301–324. 2.1.1

Turgut, A. E., Çelikkanat, H., Gökçe, F., and Şahin, E. (2008a). Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2–4):97–120. 2.1.3

Turgut, A. E., Huepe, C., Çelikkanat, H., Gökçe, F., and Şahin, E. (2008b). Modeling phase transition in self-organized mobile robot flocks. In *Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence, ANTS 2008*, volume 5217 of *Lecture Notes in Computer Science*, pages 108–119. Springer, Berlin, Heidelberg. 2.1.2

Turing, A. (1953). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society (part B)*, 237:37–72. 2.1.3

Valentini, G., Birattari, M., and Dorigo, M. (2012). Majority rule with fifferential latency: An absorbing markov chain to model consensus. In *Proceedings of the 12th European Conference on Complex Systems (ECCS)*. In press. 4.1, 4.1.2

Varghese, B. and McKee, G. (2009). A review and implementation of swarm pattern formation and transformation models. *International Journal of Intelligent Computing and Cybernetics*, 2(4):786–817. 2.1.3

Vaughan, R. T. (2008). Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2–4):189–208. 2.1.2

Visser, W., Havelund, K., Brat, G., Park, S., and Lerda, F. (2003). Model checking programs. *Automated Software Engineering*, 10(2):203–232. 5

Waibel, M., Keller, L., and Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3):648–660. 2.1.1

Wang, B., Lim, H. B., and Ma, D. (2009). A survey of movement strategies for improving network coverage in wireless sensor networks. *Computer Communications*, 32(13–14):1427–1436. 2.1.3

Wawerla, J., Sukhatme, G. S., and Matari, M. J. (2002). Collective construction with multiple robots. In *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS*, pages 2696–2701. 2.1.3

Werfel, J. (2006). Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21:20–28. 2.1.3

Werfel, J. and Nagpal, R. (2008). Three-dimensional construction with mobile robots and modular blocks. *International Journal of Robotics Research*, 27(3-4):463–479. 2.1.3

Werfel, J., Petersen, K., and Nagpal, R. (2011). Distributed multi-robot algorithms for the TERMES 3D collective construction system. In *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2.1.3

Wessnitzer, J. and Melhuish, C. (2003). Collective decision-making and behaviour transitions in distributed ad hoc wireless networks of mobile robots: Targethunting. In *Advances in Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, pages 893–902. Springer, Berlin, Heidelberg. 2.1.3

Winfield, A. F. T. (2009). Towards an engineering science of robot foraging. In *Distributed Autonomous Robotic Systems 8*, pages 185–192. Springer, Berlin, Heidelberg. 2.1.4

Winfield, A. F. T., Harper, C. J., and Nembrini, J. (2004). Towards dependable swarms and a new discipline of swarm engineering. In *Proceedings of the Internation Workshop on Simulation of Adaptive Behavior, SAB 2004*, volume 3342 of *Lecture notes in computer science*, pages 126–142. Springer, Berlin, Heidelberg. 2.1

Winfield, A. F. T., Liu, W., Nembrini, J., and Martinoli, A. (2008). Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2(2–4):241–266. 2.1.2

Wolpert, D. H. and Tumer, K. (1999). An introduction to collective intelligence. Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center. 2.1.1

Wooldridge, M. and Jennings, N. R. (1998). Pitfalls of agent-oriented development. In *Proceedings of the second international conference on Autonomous agents*, pages 385–391. ACM Press. 1

Yun, S., Schwager, M., and Rus, D. (2009). Coordinating construction of truss structures using distributed equal-mass partitioning. In *Proceedings of the International Symposium on Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 607–623. Springer, Berlin, Heidelberg. 2.1.3

Zambonelli, F., Jennings, N., and Wooldridge, M. (2001). Organisational abstractions for the analysis and design of multi-agent systems. In *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 407–422. Springer, Berlin, Heidelberg. 1

Zarzhitsky, D., Spears, D., Thayer, D., and Spears, W. (2005). Agent-based chemical plume tracing using fluid dynamics. In Hinchey, M., Rash, J., Truszkowski, W., and Rouff, C., editors, *Formal Approaches to Agent-Based Systems*, volume 3228 of *Lecture Notes in Computer Science*, pages 146–160. Springer, Berlin, Heidelberg. 2.1.2