Robot Swarms as Hybrid Systems: Modeling and Verification

Stefan Schupp
RWTH Aachen University
Aachen, Germany

Francesco Leofante
RWTH Aachen University
Aachen, Germany

Erika Ábrahám
RWTH Aachen University
Aachen, Germany

Armando Tacchella
University of Genoa
Genoa, Italy

Robot swarms can perform cooperative tasks without centralized coordination. On the one hand, decentralized control enables scalable solutions, however designing controllers for single robots that guarantee some desired global behavior of the swarm is difficult. In this work we analyse on a case study the suitability of hybrid automata to model robot swarms, and the applicability of verification methods for hybrid systems for their analysis.

1 Introduction

Robot swarms are distributed autonomous systems wherein teams of robots cooperatively perform a task, without any centralized coordination [10]. Despite relatively simple reactive controllers for individual robots, robot swarms might show complex behaviors, allowing the team to achieve goals that would defy each single robot, or would require more expensive robots [2].

While the behavior of individual robots is usually easy to understand, predicting the overall swarm behavior is difficult, and the synthesis of controllers implementing a desired swarm behavior is not straightforward. Traditional simulation-based testing approaches for swarm analysis [7, 8] suffer from intrinsically incomplete coverage. Rigorous analysis can be obtained via *formal methods* [3, 6, 11], however, most works in this direction abstract away details about the *continuous dynamics* of the robots, which may be crucial for the emergence of unforeseen behaviors. In this paper, we report on our experiences on the formal analysis of more expressive swarm models based on the discrete-continuous formalism of *hybrid automata* [5] and flowpipe-construction-based reachability analysis techniques – see [4] for an overview.

2 Swarm Engineering and Formal Methods

Despite intense research, robot swarms are still largely confined to academic research prototypes. Practical applications require confidence in the correct system behavior before deployment. In robotics, this is more than just an abstract engineering principle, as the implements can damage the environment, and thus should be subject to stringent requirements – see, *e.g.*, ISO/TC 299. Therefore, engineering swarm robotics systems should (i) get a swarm to perform a desired task and (ii) make sure that repeatable and reliable behavior can be obtained with sufficient confidence. While requirement (i) has been the subject of extensive research – see [2] for a recent review – requirement (ii) is still not considered mainstream. Following standard practice, first the system requirements are specified before a design respecting those

Submitted to:
SNR 2018

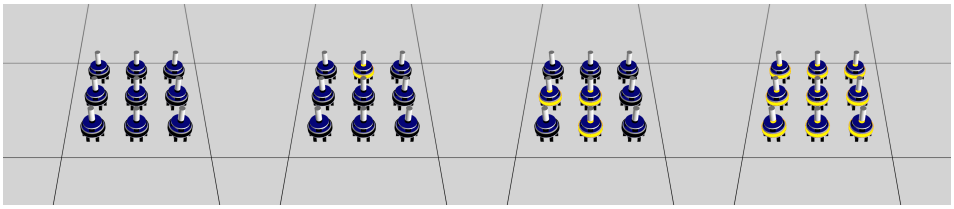


Figure 1: Synchronization in the simulator.

requirements is produced, implemented and tested against the requirements. As single robots are embedded systems, for this procedure state-of-the-art methodologies like model-driven engineering [13] can be considered for single-robot requirements, but these methodologies cannot tackle swarm-level requirements.

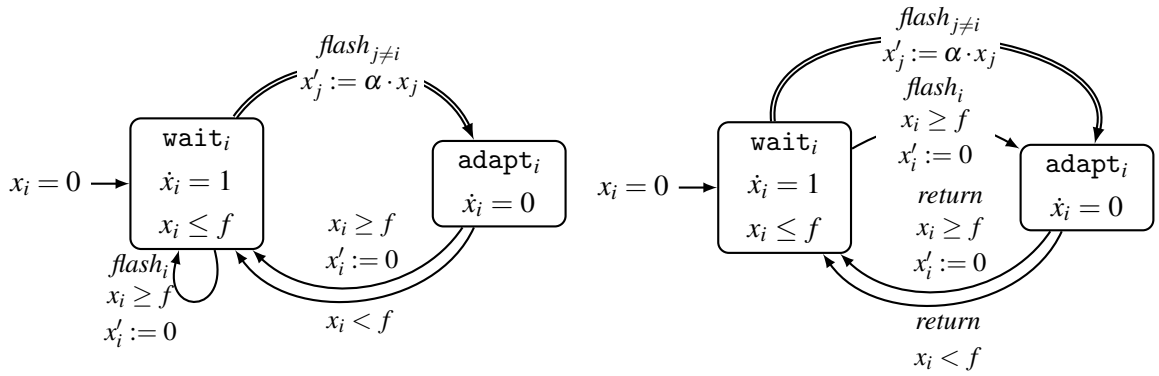
As traditional system engineering techniques may not be suitable for swarm robotics, *formal verification* could have an edge. To the best of our knowledge, the first contribution along this line is [12], wherein the authors investigated the verification and validation of spacecraft using swarm technology. A number of approaches was considered, including process algebras, X-machines and Unity Logic. However, at the time of the contribution (2004), the conclusion was that none of the approaches had all the properties required to assure correct behavior and interactions of swarms in the context of the ANTS (Autonomous Nano Technology Swarm) concept mission. A series of papers by Dixon et al. – see, *e.g.*, [6] for the most recent contribution in the series – explores probabilistic models and verification thereof to prove swarm-level requirements. Noticeably, in the case of probabilistic models and logic, the model of each single robot controller is very close to the actual implementation, and model checking of relevant properties is reported to be feasible for small swarms only – less than 4 robots according to the experiments in [6]. Finally, [3] applies probabilistic finite state machines and probabilistic temporal logic to provide property-based design of swarm robotic systems.

All the above approaches neglect the dynamics of the robots and their interactions with the environment. For swarms, this may hamper our ability to determine whether they always behave correctly. Indeed, all robotic systems involve programmed digital controllers interfacing with the physical world. Therefore, an accurate model of robot operation should include both the (discrete) control states and the (continuously) varying physical quantities.

3 Modeling Swarms as Hybrid Automata – An Example

In the following, we discuss the modeling of swarms as *hybrid automata* [5], a well-established formalism to model systems combining discrete and continuous dynamics.

In this work we reproduce the behavior of pulse-coupled oscillators as described in [9] and implemented in the ARGoS simulation environment (see Fig. 1). The model involves a population of n MarXbots [1], each of which is equipped with an LED. For $i \in \{1,\dots,n\}$, the ith robot is characterized by a clock x_i subject to the continuous dynamics $\dot{x}_i = 1$, which applies as long as $0 \leq x_i \leq f$ for some *firing threshold* $f \in \mathbb{R}_{>0}$. When $x_i = f$, robot i flashes its central LED and x_i is reset to zero by a discrete event. Robots interact by a simple form of pulse coupling: when robot i flashes, all other robots are



(a) First instance of model using label synchronization (`lsync I`).

(b) Refined instance using label synchronization with synchronized return to wait (`lsync II`).

Figure 2: Models of one robot in the synchronization benchmark using label synchronization; all jumps are urgent, jumps of similar shape are collected and depicted as double lines.

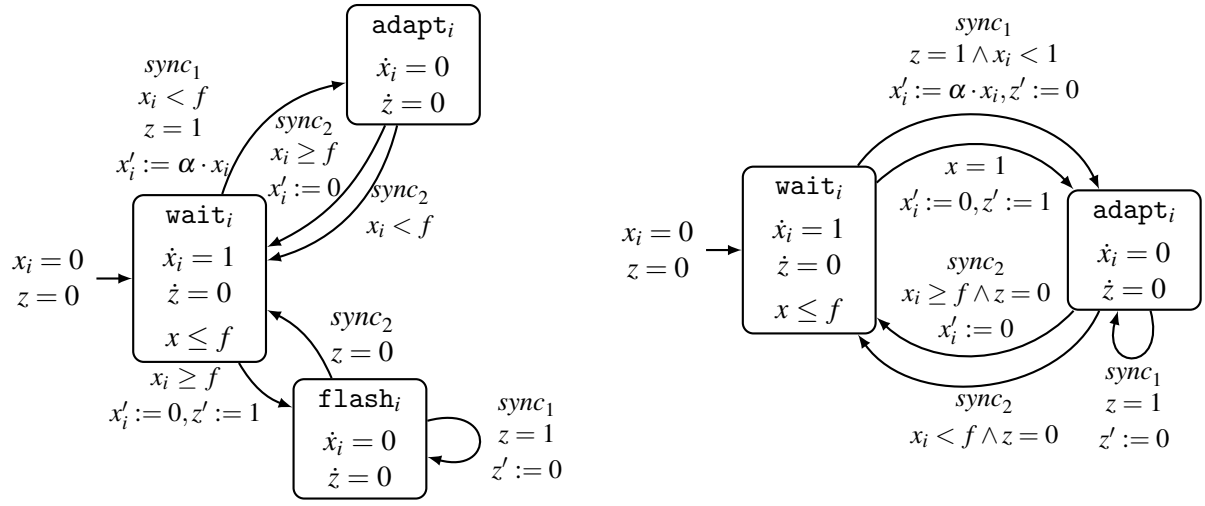
pulled towards firing according to the following relation for some $\alpha \in \mathbb{R}_{>1}$:

$$x_i = f \implies x_j := \begin{cases} \alpha \cdot x_j & \text{if } \alpha \cdot x_j < f \\ 0 & \text{otherwise} \end{cases} \qquad \textit{for all } j \in \{1,\ldots,N\} \setminus \{i\} \tag{1}$$

Note that Eq. 1 only describes the update of the clocks – neither the flashing nor implicit flashing of a robots' LED whenever the clock is reset to 0 is described. These properties have to be added to the model to make them observable. Despite the simple model, the problem of pulse coupling represents a good example of how global swarm behaviors can emerge in distributed systems without being explicitly specified by individual control algorithms. Indeed, a global synchronization of flashing behaviors is achieved – *i.e.*, all clocks are synchronized – even though this is not explicitly imposed by individual controllers.

Modeling. We propose several approaches on how to model the synchronization problem as a compositional system. Each robot is modeled by a hybrid automaton $\mathscr{H}_i$ such that the swarm behavior for a swarm of size n is modeled by a hybrid automaton $\mathscr{H}$ obtained by parallel composition $\mathscr{H} = \mathscr{H}_1 || \ \ldots || \mathscr{H}_n$ of the single components. In the following we will discuss the different approaches in detail. Note that all transitions in our models are *urgent*, which forces the control to take the respective transition as soon as its guard condition is satisfied. All approaches model the case distinction of Eq. 1 via guarded discrete transitions to locations used to reflect the clock updates. Note that all approaches use constructions of urgent transitions, labels and guards which implicitly guarantee that no time passes in those locations required for adaption but only in the location `wait`.

Label synchronization. Modeling compositional hybrid systems involving communication between the components can be achieved by annotating transitions in the single components by a *synchronization label*. Labeled transitions in one component can only be taken, whenever for each other component it is possible to take a transition annotated with the same label at the same time – in this case all components synchronize on this transition. In our case to model flashing we can create a set of labels flash_i, one for each component. The resulting automaton for a single component can be found in Fig. 2a. In case the clock valuation of a robot r_i reaches f, in the respective automaton a transition with label flash_i is enabled. At the same time all other components take a synchronizing transition to the location `adapt` used to model the clock updates.



(a) Model, 1st version (`shd I`).

(b) Reduced model (`shd II`).

Figure 3: Hybrid automata modeling a single robot in the synchronization benchmark using a shared variable z for synchronization.

The syntax and semantics of hybrid automata do not allow for multiple synchronization labels on a single transition. Therefore we add a synchronizing transition from `wait` to `adapt` for each other component. While the model for a single component is relatively simple, using several synchronizing transitions has a strong impact on the outcome of the parallel composition of several components, as the number of transitions drastically increases with the number of modeled robots (see Tab. 2). Furthermore a missing synchronization on return to the location `wait` adds unnecessary complexity. In the parallel composition of n components of `lsync I` all possible sequences of single components returning to location `wait` are encoded. In this setup all those sequences are built from urgent transitions where additionally the order of execution does not matter. Applying *partial order reduction* by introducing a fixed order of execution already can reduce the resulting automaton, however in this special case we can even collect all transitions into one single transition which results in an improved version (`lsync II`). Note that in general equivalence needs to be shown when unifying sequences of transitions – in our case we know the execution order does not matter and no time passes in between the single jumps. The same result can also be obtained when we allow for synchronization on returning to location `wait` (see Fig. 2b).

Even though it is natural to use label synchronization for this task, this way of modeling has the drawback that each component requires full information about the total number of components (and their respective synchronization labels), which does not allow for a generic approach. Furthermore, there is no way to observe directly that two components flash at the same time, as the synchronization labels mutually exclude each other and thus there is no system state which directly indicates synchronization.

Shared variables. To present an approach where single components do not require any prior knowledge about the full system we present two versions using a shared variable z to overcome multiple synchronization labels. The first approach (see Fig. 3a) uses two additional locations per component – one for adaption of other variables and one to prepare the synchronized jumps. The synchronization labels in both versions are optional – in fact they help to reduce the number of transitions in the resulting parallel composition similar to the previous approach. Signaling of a flash is done via the shared variable z

Table 1: Number of locations and transitions in the resulting automata for different numbers of robots.

		# robots							
	version	1	2	3	4	5	6	7	8
#locs.	`shd I`	3	7	15	31	63	127	255	511
	`shd II`	2	4	8	16	32	64	128	256
	`lsync I + II`	2	3	7	15	31	63	127	255
#trans.	`shd I`	6	18	54	162	486	1458	4374	13122
	`shd II`	5	13	35	97	275	793	2315	6817
	`lsync I`	3	6	33	164	755	3310	14077	58728
	`lsync II`	3	6	21	68	215	670	2065	6313

which, whenever one clock reaches the threshold is set to 1, thus enabling the according transitions in each component. In contrast to label synchronization, one additional location is required to be able to set $z = 1$, which acts as a synchronization flag. A reduced version (`shd II`, see Fig. 3b) uses similar mechanisms but unifies the locations `adapt` for adaption and `flash` for setting the synchronization flag into one location to reduce the parallel composition result.

3.1 Results

The introduction of additional locations where it is ensured that no time passes in theory does not increase the complexity of the analysis as no flow occurs. However, in practice especially guarded (urgent) transitions between those locations heavily influence running times of analysis methods, as each guard condition has to be verified. Our presented approaches employ urgency and *multi-edges*, *i.e.*, two or more transitions connect the same pair of locations, a feature which is not supported by all analysis tools. Other modeling choices could have been made, *e.g.*, eliminating multi-edges by introducing additional intermediate discrete locations with exclusive transition guards, or enforcing urgency with a combination of invariants and guards. However this would have resulted in an unnecessarily complex model.

To compare our approaches we have created models for the synchronization benchmark for up to 8 robots using all presented approaches. Statistics about the resulting hybrid automata can be found in Tab. 2. From our results we can observe, that the natural approach via label synchronization (`lsync I`) creates a high number of transitions in the resulting composed automaton while keeping the number of locations low. Applying the optimization (`lsync II`) which effectively collects sequences of urgent transitions into a single urgent transition, reduces the number of transitions drastically. The versions using shared variables (`shd I + II`) produce results with less transitions, *e.g.* edges for the return to `wait` are collected by a synchronization label ($sync_2$) and thus create one jump in the parallel composition.

Reachability analysis. Figure 4 shows flowpipes for a system with 3 robots with parameters $f = 100, \alpha = 1.1$ computed with our reachability analysis tool HyDRA, which is based on the `C++` library for state set representations HyPro [14]. All robots start with different initial clock valuations ($x_1 \in [0,5], x_2 \in [15,20], x_3 \in [30,35]$). We use a local time horizon (max. time spent in one location) of 110 sec. and support functions as a state set representation with a time step size of 0.01 sec.. In the enlarged part (right) we can observe the first sequence of flashes. Robot r_3 reaches the firing threshold first and flashes, which causes the other robots to adapt their clock values. After that robot r_2 is the next one to

Table 2: Running times for different numbers of robots using $f = 1, \alpha = 1.1$ (time step size: 0.01, jump depth: 20, state set representation: boxes). Timeout (TO) was set to 10 minutes.

	# robots							
version	1	2	3	4	5	6	7	8
`shd I`	**0.11**	**0.11**	**0.12**	0.16	0.48	9.88	TO	TO
`shd II`	**0.11**	**0.11**	0.13	0.19	0.75	16.85	TO	TO
`lsync I`	0.12	**0.11**	0.13	0.21	1.02	64.6	TO	TO
`lsync II`	0.12	**0.11**	**0.12**	**0.14**	**0.25**	**2.96**	**146**	TO

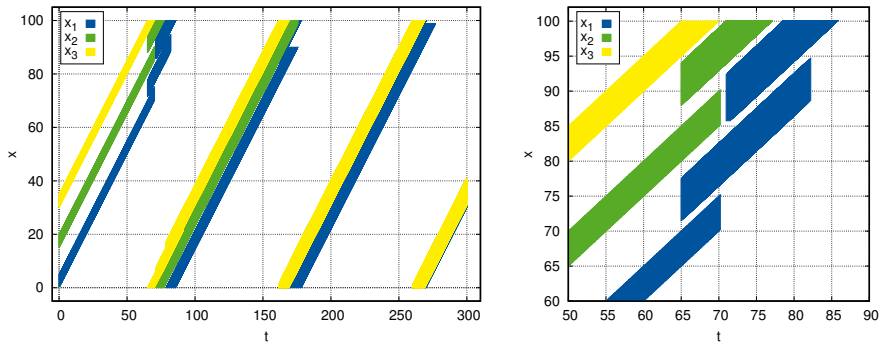


Figure 4: Overlayed flowpipes for a system of 3 robots for $f = 100, \alpha = 1.1$ (local time horizon 110 sec.). Right: excerpt showing clock adaption after flashing.

flash, which results in a second adaption for robot r_1. After the third flash after 250 sec. of robot r_3 all cycles are approximately synchronized.

4 Conclusion and Future Directions.

We have presented several approaches towards modeling a swarm of robots implementing synchronization behaviors without centralized coordination. Our experimental analysis show that modeling already a simple system like this poses several challenges which need to be addressed to be able to apply and scale formal verification techniques to high-dimensional swarms. This opens up new perspectives for researchers to devise clever abstractions and techniques to represent complex systems like robotics swarms. As shown in this paper, simple modeling choices can speed up computations making a first step towards the applicability of reachability based techniques to robot swarm.

References

[1] Michael Bonani, Valentin Longchamp, Stéphane Magnenat, Philippe Rétornaz, Daniel Burnier, Gilles Roulet, Florian Vaussard, Hannes Bleuler & Francesco Mondada (2010): *The MarXbot, a miniature mobile robot opening new perspectives for the collective-robotic research*. In: *Proc. of IROS'10*, pp. 4187–4193.

[2] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari & Marco Dorigo (2013): *Swarm robotics: a review from the swarm engineering perspective*. *Swarm Intelligence* 7(1), pp. 1–41.

[3] Manuele Brambilla, Carlo Pinciroli, Mauro Birattari & Marco Dorigo (2012): *Property-driven design for swarm robotics*. In: *Proc. of AAMAS'12*, pp. 139–146.

[4] Colas Le Guernic (2009): *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. Ph.D. thesis, Joseph Fourier University, Grenoble, France.

[5] Thomas A. Henzinger (1996): *The Theory of Hybrid Automata*. In: *Proc. of LICS'96*, IEEE Computer Society Press, pp. 278–292.

[6] Savas Konur, Clare Dixon & Michael Fisher (2012): *Analysing robot swarm behaviour via probabilistic model checking*. *Robotics and Autonomous Systems* 60(2), pp. 199–213.

[7] Thomas Halva Labella, Marco Dorigo & Jean-Louis Deneubourg (2004): *Efficiency and Task Allocation in Prey Retrieval*. In: *Proc. of BioADIT'04*, pp. 274–289.

[8] Wenguo Liu, Alan F. T. Winfield, Jin Sa, Jie Chen & LiHua Dou (2006): *Strategies for Energy Optimisation in a Swarm of Foraging Robots*. In: *Proc. of SAB'06*, pp. 14–26.

[9] Renato E Mirollo & Steven H Strogatz (1990): *Synchronization of pulse-coupled biological oscillators*. *SIAM Journal on Applied Mathematics* 50(6), pp. 1645–1662.

[10] Lynne E. Parker (2000): *Current State of the Art in Distributed Autnomous Mobile Robotics*. In: *Proc. of DARS'00*, pp. 3–14.

[11] Joaquín Peña, Christopher A. Rouff, Mike Hinchey & Antonio Ruiz Cortés (2011): *Modeling NASA swarm-based systems: using agent-oriented software engineering and formal methods*. *Software and System Modeling* 10(1), pp. 55–62.

[12] Christopher Rouff, Amy Vanderbilt, Michael G. Hinchey, Walt Truszkowski & James L. Rash (2004): *Properties of a Formal Method for Prediction of Emergent Behaviors in Swarm-Based Systems*. In: *Proc. of SEFM'04*, pp. 24–33.

[13] Douglas C Schmidt (2006): *Model-driven engineering*. *IEEE Computer Society* 39(2), p. 25.

[14] Stefan Schupp, Erika Ábrahám, Ibtissem Ben Makhlouf & Stefan Kowalewski (2017): *HyPro: A C++ Library of State Set Representations for Hybrid Systems Reachability Analysis*. In: *Proc. of NFM'18*, pp. 288–294.