Towards Temporal Verification of Emergent Behaviours in Swarm Robotic Systems

Clare Dixon[1], Alan Winfield[2], and Michael Fisher[1]

[1] Department of Computer Science, University of Liverpool, Liverpool, UK
{cldixon,mfisher}@liverpool.ac.uk
[2] Bristol Robotics Laboratory, University of the West of England, Bristol, UK
Alan.Winfield@uwe.ac.uk

Abstract. A robot swarm is a collection of simple robots designed to work together to carry out some task. Such swarms rely on: the simplicity of the individual robots; the fault tolerance inherent in having a large population of often identical robots; and the self-organised behaviour of the swarm as a whole. Although robot swarms are being deployed in increasingly sophisticated areas, designing individual control algorithms that can *guarantee* the required global behaviour is difficult. In this paper we apply and assess the use of *formal verification* techniques, in particular that of model checking, for analysing the emergent behaviours of robotic swarms. These techniques, based on the automated analysis of systems using *temporal logics*, allow us to analyse all possible behaviours and so identify potential problems with the robot swarm conforming to some required global behaviour. To show this approach we target a particular swarm control algorithm, and show how automated temporal analysis can help to refine and analyse such an algorithm.

1 Introduction

The use of autonomous robots has become increasing appealing in areas which are hostile to humans such as underwater environments, contaminated areas, or space [24,2,21]. Rather than deploying one or two, often large and expensive, robots a significant focus is now on the development of *swarms* of robots.

A robot swarm is a collection of simple (and usually identical) robots working together to carry out some task [3,18]. Each robot has a relatively small set of behaviours and is typically able to interact with other (nearby) robots and with its environment. Robot swarms are particularly appealing in comparison to one or two more complex robots, in that it may be possible to design a swarm so that the failure of some of the robots will not jeopardize the overall mission, i.e. the swarm is *fault tolerant*. Such swarms are also advantageous from a financial point of view since each robot is very simple and mass production can significantly reduce the fabrication costs.

Despite the advantages of deploying swarms in practice, it is non-trivial for designers to formulate individual robot behaviours so that the emergent behaviour of the swarm as a whole is guaranteed to achieve the global task of the swarm,

R. Groß et al. (Eds.): TAROS 2011, LNAI 6856, pp. 336–347, 2011.

and that the swarm does not exhibit any other, undesirable, behaviours [19]. Specifically, it is often difficult to predict the overall behaviour of the swarm just given the local robot control algorithms. This is, of course, essential if swarm designers are to be able to effectively and confidently develop reliable swarms. So, we require some mechanism for analysing what the swarm can do, given the behaviour of individual robots and a description of their possible interactions both with each other and with the environment. Using this mechanism, the designer can assess to what extent the swarm behaves as required and, where necessary, redesign to avoid unwanted outcomes.

Currently, the analysis of swarm behaviour is typically carried out by experimenting with real robot swarms or by simulating robot swarms and testing various scenarios (eg see [20,17]). In both these cases any errors found will only be relevant to the particular scenarios constructed; neither provides a comprehensive analysis of the swarm behaviour in a wide range of possible circumstances. Specifically, neither approach can detect a problem where undesirable behaviour occurs in some untested situation. We note that approaches to modelling robot swarms have been proposed in, for example, [15,16,10].

A well-known alternative to simulation and testing is to use *formal verification*, and particularly the technique called *model-checking* [6]. Here a mathematical model of *all* the possible behaviours of the system is constructed and then all possible executions through this model are assessed against a required logical formula representing a desired property of the system. In the case of systems, such as robot swarms, the mathematical model usually represents an abstraction of the real control system, while the logical formula assessed is usually a *temporal* formula representing the presence of a desirable property or the absence of an undesirable property on all paths; see, for example [9].

In this paper we will explore the use of temporal verification for robot swarms in an effort to formally verify whether such swarms do indeed exhibit the required global behaviour. The structure of this paper is as follows. In Section 2 we give details of the temporal logic in which logical properties we wish to show are given and provide an overview to model checking. In Section 3 we describe our use of formal verification to assess swarm algorithms and introduce one existing algorithm, namely Nembrini's *alpha algorithm* [17], that we will analyse. In Section 4 we describe the abstractions we use in more detail. In Section 5 we give verification results from using the temporal model checker and the impact of these results both on the original algorithm and the abstractions. In Section 6 we discuss the results and we provide concluding remarks in Section 7.

2 Temporal Logic and Model Checking

The logic we consider is propositional linear-time temporal logic, called PTL where the underlying model of time is isomorphic to the Natural Numbers, $\mathbb{N}$. A model for PTL formulae can be characterised as a sequence of *states* of the form: $\sigma = s_0, s_1, s_2, s_3, \ldots$ where each state, s_i, is a set of proposition symbols, representing those propositions which are satisfied in the i^{th} moment in time.

In this paper we will only make use two of temporal operators '$\Diamond$' (*sometime in the future*), '$\Box$' (*always in the future*) in our temporal formulae. The notation $(\sigma, i) \models A$ denotes the truth of formula A in the model σ at state index $i \in \mathbb{N}$ defined as follows where PROP is a set of propositional symbols.

$$\begin{array}{ll} (\sigma, i) \models p & \text{iff } p \in s_i \text{ where } p \in \text{PROP} \\ (\sigma, i) \models \Diamond A & \text{iff } \exists k \in \mathbb{N}.\ (k \geqslant i) \text{ and } (\sigma, k) \models A \\ (\sigma, i) \models \Box A & \text{iff } \forall k \in \mathbb{N}.\ (k \geqslant i) \text{ and } (\sigma, k) \models A \end{array}$$

Model checking [6] is a popular technique for verifying the temporal properties of systems. Input to the model checker is a model of the paths through a system (a finite-state transition system) and a formula to be checked on that model. The language of the formula to be checked is usually some form of temporal logic for example the linear-time temporal logic, PTL, or the branching-time temporal logic, CTL. A number of model checkers have been developed but we will use `NuSMV` [5] which allows properties expressed in both CTL and PTL.

Essentially, we construct a set of finite-state transition systems, corresponding to each of the robots in the swarm, and then model-check a PTL formula against the concurrent composition of these transition systems. The key element of model-checkers is that, if there are execution paths of the system that *do not* satisfy the required temporal formula, then at least one such "failing" path will be returned as a counter-example. If no such counter-examples are produced then all paths through the system indeed satisfy the prescribed temporal formula.

3 Analysing Swarm Algorithms

Our long term aim is to develop, deploy and extend formal verification techniques for use in swarm robotics and so show that formal verification is viable in this context. Yet, even within swarm robotics, the application of formal verification is difficult. The continuous aspects of both the robotic control system and the robots' movements and environment do not sit well with the discrete and finite nature of model-checking. Fortunately, in developing simple robotic control algorithms, engineers typically use finite-state machines as part of their behavioural design. Thus, we can base our verification on such finite-state machines provided by roboticists. There remains the problem of the use of continuous functions/variables in such state machines. As in the verification of *hybrid systems*, we must provide abstractions to simplify such continuous values so that model-checking can be carried out [11]. Finally we note that even with such abstractions, representing the location and movement for a number of robots will generate a huge state space so initially we must focus on small grid sizes and numbers or robots. Thus, in summary, our approach is as follows.

1. Take the design of a swarm control algorithm for an individual robot, as represented in the form of a finite-state machine.
2. Describe an abstraction that tackles the continuous nature of the domain, the potentially large number of robots, and the nature of concurrent activity and communication within the swarm.

3. Carry out (automatic) model checking to assess the temporal behaviour of the model from (2). If model-checking succeeds, then return to (2) refining the abstraction to make it increasingly realistic. If model-checking fails, returning a scenario in which the temporal requirement is not achieved, then analyse (by hand) how the algorithm in (1) *should* cope with this scenario. Either there is a problem with the original algorithm, so this must be revised, or the algorithm is correct for this scenario and so the abstraction in (2) must be revisited and expanded to capture this behaviour.

This process is continued until no errors are found in (3) and the abstraction in (2) is sufficiently close to the physical scenario to be convincing. While this cycle is clearly not (and cannot be) fully automatic, the results from model-checking help direct us in refining the algorithm and/or abstractions used in the design. This approach follows the spirit of the "counter-example guided abstraction refinement" method initiated in [8] and applied to hybrid systems in [1].

3.1 The 'Alpha' Algorithm

As a case study we consider algorithms for robot swarms which make use of local wireless connectivity information alone to achieve swarm aggregation. Specifically, we examine the simplest (alpha) algorithm described in [17,22]. Here, each robot has range-limited wireless communication which, for simplicity, we model as covering a finite distance in all directions from the robot's location. Beyond this boundary, robots are out of detection range. The basic alpha algorithm is very simple:

- The default behaviour of a robot is forward motion.
- While moving each robot periodically sends an "Are you there?" message. It will receive "Yes, I am here" messages only from those robots that are in range, namely its neighbours.
- If the number of a robot's neighbours should fall below the threshold α then it assumes it is moving *out* of the swarm and will execute a 180° turn.
- When the number of neighbours rises above α (when the swarm is regained) the robot then executes a random turn. This is to avoid the swarm simply collapsing in on itself.

Thus, we assume that each robot has three basic behaviours: move forward (default); avoidance (triggered by the collision sensor); and coherence (triggered by the number of neighbours falling below α).

4 Abstraction

In the following we explain, in more detail, the abstractions that are used to build an appropriate model to be checked.

Spatial Aspects. We consider a number of identical robots moving about a square grid and assume that the grid is divided into squares with at most one robot in each square. We assume a step size of one grid square and that a robot can detect other robots for purposes of avoidance in the adjacent squares. The robot has a direction it is moving in. Rather than taking a bearing, we describe this as one of *North*, *South*, *East* or *West*. To make the problem finite we limit the grid size to be an $n \times n$ square which wraps round, i.e. a movement North from the upper edge of the grid will result is a move to the lower edge of the grid. Initially the robots may have any direction but are placed on the grid where they are connected but in different grid squares. Initially any robot may move first.

Connectivity. Regarding connectivity, this is calculated from the robots' relative positions. Initially we assume that each robot can detect other robots in the eight squares surrounding it. Hence if a robot is in square (1,1) it can detect robots in squares (0,0), (0,1), (0,2), (1,0), (1,2), (2,0), (2,1), and (2,2). Thus it has a wireless range of one square in all directions. Initially we set the value of $\alpha = 1$ i.e. a robot is connected if it can detect at least one other robot. In our verification, we aim to show that for all i, $\Box\Diamond con_i$ follows from our specification, i.e. each robot stays connected infinitely often. Thus, the swarm need not *always* be connected but, if it becomes disconnected, should eventually reconnect. In particular, no specific robot will remain disconnected forever.

Motion Modes. Avoidance is dealt with as follows. If a robot is moving in some direction and the square ahead is occupied: move to the right or left; if these are both occupied move backwards; else stay in the current position. The original direction the robot was moving in is maintained. Each robot can be in one of two motion modes: *forward* or *coherence*. The connectivity of each robot can also be in one of two modes: *connected* or *not connected*. The combination of motion and connectivity give us four possible alternatives.

- In the forward mode, when connected, move forward and the motion mode remains 'forward'.
- In the forward mode, but not connected, turn 180° and change the motion mode to 'coherent'.
- In the coherent mode, but not connected, move forward and the motion mode remains as 'coherent'.
- In the coherent mode, when connected, perform a 90° turn (i.e. either 90° left or 90° right) and change the motion mode to 'forward'.

Concurrency. An important variety of abstraction concerns the representation of concurrent activity within the swarm of robots. Thus, in modelling this aspect we must ask questions about whether all robots run simultaneously, whether some robots run faster than others, etc. Thus, there is a wide variety of different abstractions we might use, which in turn correspond to different mechanisms for concurrently composing the robot transition systems/models.

- *synchrony* — all robots execute at the same time and with the same clock.
- *(strict) turn taking* — execution of the robots is essentially interleaved, but the robots must execute in a certain order, e.g. r_1, r_2, r_3, r_1, r_2, r_3, etc.
- *(non-strict) turn taking* — execution of the robots is again interleaved but for m robots in every cycle of m steps each robot moves once, so we can now have a situation where a robot executes two steps consecutively, e.g. r_1, r_2, r_3, r_3, r_2, r_1, etc.
- *(fair) asynchrony* — robots execute at the same time, yet some robots are faster than others. However the *fair* aspect ensures that a robot can only take a finite number of steps before all other robots have finished their step.

It is important to note that the particular view of concurrency taken can significantly affect the results.

5 Verification Using Model Checking

We model Nembrini's alpha algorithm and aim to verify $\Box\Diamond con_i$ for each of the i robots. The model is a finite-state transition system written in `NuSMV`'s input language representing the algorithm in Section 4. Due to the large state spaces involved we assume initially that the grid is 5×5 and there are two robots and increase these values. We appreciate both the grid size and number of robots is small but this will increase while allowing the grid to wrap round means that the robots can, for example, move North forever. As well as considering different number of robots and grid sizes we will change the abstraction we use relating to concurrency, α parameter and wireless range.

5.1 Results: Concurrency

First we consider the different concurrency options: fair asynchrony, non-strict turn taking, strict turn taking and then synchrony. In the table below we show some results of running `NuSMV` on the input files for a number of different sized input grids with two or three robots. Connectedness is a global property calculated from the robots' positions. Note that a response of '*true*' from the model checker means that every path from every initial state of the model satisfies this property. A response of '*false*' means not all paths satisfy the property and `NuSMV` outputs a failing trace.

Problem	`NuSMV` Output			
	Fair Async.	Non-Strict	Strict	Sync.
5×5 grid, 2 robots	false	false	false	true
6×6 grid, 2 robots	false	false	false	true
7×7 grid, 2 robots	false	false	false	true
8×8 grid, 2 robots	false	false	false	true
5×5 grid, 3 robots	false	false	false	false
6×6 grid, 3 robots	false	false	false	false

First considering fair asynchrony we can obtain failing traces for all grid sizes and number of robots we tried. Considering the failing trace for the grids just with two robots, after some initial moves the two robots both get in the coherent non-connected mode, moving at right angles to each other and remain disconnected for ever.

Similarly for strict and non-strict turn taking the table shows that `NuSMV` can show that the property does not occur on all paths from every initial state, size of grid and number of robots we have experimented with. Looking at the failing traces for non-strict turn taking, robot one makes a move resulting in the robots losing connectedness which they never regain. This may be due to a number of reasons. Firstly robot one has to wait two steps before it can take any action relating to this loss of connectedness (recall that the other robot can move twice). This might be avoided by using a strict turn taking abstraction. However, as can be seen in the table, experiments with strict turn taking also give a result of *false* in the same cases as previously. Similar to the above, the failing traces show a loss of connection that is never regained. These results show that changing the computational abstraction (from non-strict to strict) does not, in itself, solve our failure.

However changing the concurrency mode to synchronous we now obtain '*true*' results for *all* the two robot cases but again for all the three robot cases tried we obtain '*false*'. Observing the successful traces of the two robot cases, if the robots are both moving in the same direction originally they continue in this same direction in the forward connected mode forever. If they are in different directions they move away from each other, correct this in the coherent mode by moving back together become re-connected and again repeat one of the above two patterns again. This leads us to believe that synchrony is probably more appropriate for modelling the alpha algorithm.

State	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	1.11
Loc r_1	(0,0)	(4,0)	(3,0)	(2,0)	(1,0)	(0,0)	(4,0)	(3,0)	(2,0)	(1,0)	(0,0)
Dir r_1	W	W	W	W	W	W	W	W	W	W	W
Mode r_1	FC	FC	FC	FC	FC	FC	FC	FC	FC	FC	FC
Loc r_2	(1,1)	(0,1)	(4,1)	(3,1)	(2,1)	(1,1)	(0,1)	(4,1)	(3,1)	(2,1)	(1,1)
Dir r_2	W	W	W	W	W	W	W	W	W	W	W
Mode r_2	FC	FC	FC	FC	FC	FC	FC	FC	FC	FC	FC
Loc r_3	(4,1)	(4,2)	(4,3)	(4,2)	(0,2)	(4,2)	(3,2)	(2,2)	(1,2)	(0,2)	(4,2)
Dir r_3	N	N	N	S	E	W	W	W	W	W	W
Mode r_3	FC	FC	FNC	CC	FNC	CNC	CNC	CNC	CNC	CNC	CNC

Above we provide a failing trace for the synchronous 5×5 grid output for three robots produced by `NuSMV` where r_i denotes robot i. In that trace we use the following abbreviations FC-forward connected; FNC-forward and not connected; CC-coherent connected; CNC-coherent not connected; Loc-location and Dir-Direction. From state 1.6 onwards robots one and two remain in the forward connected mode travelling West whereas robot three is in the coherent non-connected mode also travelling West. State 1.11 is the same as 1.6 showing that the pair of robots can loop round the states 1.6 to 1.11 forever with robot three remaining disconnected.

It could be argued that the reasons for this are that in the three robot case we need a bigger α parameter, or that the wireless range is too small potentially

resulting in frequent loss of connection (or both). Hence we next attempt to consider these cases specifically by increasing the α parameter, i.e. in the three robot case each robot needs to remain connected to two others and then by requiring the wireless range to be larger than the step size.

5.2 Results: Changing the Alpha Parameter and Wireless Ranges

We now increase the α parameter from one to two, meaning that each robot must be able to detect two others, and also consider increasing the wireless range. In the following we assume synchronous concurrency. The results are presented in the table below. A dash in the table means that this experiment makes no sense for either an increase in the α parameter or wireless range.

Problem	NuSMV Output	
	$\alpha = 2$	Range = 2
6×6 grid, 2 robots	-	true
7×7 grid, 2 robots	-	true
8×8 grid, 2 robots	-	true
9×9 grid, 2 robots	-	true
10×10 grid, 2 robots	-	true
5×5 grid, 3 robots	false	-
6×6 grid, 3 robots	false	false
7×7 grid, 3 robots	false	false
8×8 grid, 3 robots	false	false

Regarding the increase in the α parameter we need to have at least three robots. Hence we only consider three robot cases. Regarding wireless range this was previously set at one square from a robot. We now increase this to be two squares from each robot, i.e. a robot can detect others in the 5×5 squares centred on the robot. Obviously as we have increased the detection area, in grid sizes of 5×5 and smaller all robots will always be connected at all moments. Hence we only consider grid sizes of 6×6 and above.

We note that now, with this modified alpha algorithm (i.e. with $\alpha = 2$) we still obtain results of '*false*' for all the three robot cases tried. Considering the failing traces for three robots we reach a cycle where all the robots are moving in the same direction in the coherent non-connected mode, i.e. there aren't two other robots in range. Something similar happens with the increased wireless range, i.e. we reach a cycle where all the robots move in the same direction, two within wireless range in the forward connected mode and one not within wireless range in the coherent non-connected mode. We note that we can also find a failing trace for both $\alpha = 2$ and wireless range $= 2$ for an 8×8 grid.

We can continue this until either the abstractions are realistic enough, or until our verification attempts take too much time/space. We will discuss our results in the next section.

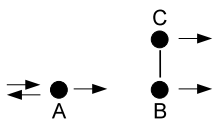
6 Discussion and Related Work

The alpha algorithm has been well studied and, to date, validated with simulated and real robots [17], and using a probabilistic mathematical modelling approach [22]. However, neither simulation, real robot experiments, nor mathematical modelling provide formal proof of the algorithm's correctness. Consider simulation (or real robot experiments, which may be regarded as 'embodied' simulations). Given that simulated (or real) robots move in a real-valued, not grid-, world with typical swarm sizes of 40 robots and α values of 5, 10 or 15, for 10,000 seconds [22], we have a practically infinite state-space and each simulation run tests only a tiny number of paths through that state-space.

As outlined in Section 4, the model checking approach within this paper attempts to formally establish correctness by reducing the state-space to a tractable size so that *every* path through that state-space can be tested. That reduction is achieved by means of introducing, firstly, a number of simplifying assumptions and, secondly, by running very small swarm sizes (2 or 3 robots) and very small grid-worlds (5×5 up to 8×8). We thus need to ask ourselves whether the failure to verify the correctness of the algorithm in some cases of Section 5 is because (a) the algorithm is flawed or (b) the simplifying assumptions (necessary for tractability) are so severe that they go beyond the bounds within which the algorithm should be expected to work.

First we consider the method of concurrency used to model the problem. We believe that fair asynchrony, non-strict and strict turn taking are unsuitable for modelling this problem because in the first two cases one robot may be able to make two moves before one of the others makes a move and in the latter case a robot has to wait for the others to make moves before it can react to a loss in connectedness. This is confirmed by the '*false*' results obtained in Section 5.1, i.e. such failing traces would not be represented by the actual runs of the algorithm. Hence we believe that synchrony is the only reasonable abstraction because for the timing of the real robots we assume that any two robots that become disconnected each notice the disconnection at approximately the same time.

However, we note that with the three robot synchronous cases we can still obtain failing traces. Some of these are of the form presented below. This illustrates a possible scenario in which robot A has become disconnected from the swarm and turns round but never catches up with B and C. Because robots B and C remain connected to each other, with $\alpha = 1$, B and C do not react to the loss of robot A and A remains disconnected.



We consider both increasing the α parameter and wireless range but still obtain failing traces similar to the above form. These results appear to confirm a known problem with the alpha algorithm, when a robot or group of robots is

linked to the rest of the swarm by a single link (known as a bridge or cutvertex). This problem was discussed by Nembrini [17] and overcome by the improved beta algorithm [17].

Regarding the *wrap round* grid abstraction we note that potentially some failing traces could not appear in actual runs of the algorithm. Take a small grid size, for example 4×4, and a disconnected robot travelling in some direction, for example West. The wrap round nature of the grid may mean the robot becomes re-connected to robots that are some distance East of the robot. For this reason we have excluded any results from 4×4 grids and have checked that the failing traces listed above do not suffer from this problem. Let the term *grid independent* mean that the robot movement in a failing trace for a wrap round grid can be translated into an infinite grid obtaining the same connectedness values for each robot. We conjecture that, in the case of synchrony, once we have found a *grid independent trace* for an $n \times n$ grid be can extend this to a *grid independent trace* for an $m \times m$ grid where $m > n$. Hence when we find one such trace for a number of synchronous robots and some grid size we do not have to try any larger sized grids.

Obviously we would like to consider larger number of robots and, even with the above observation, may need to consider larger grid sizes but we are faced by the well known state explosion problem [7]. Even with the simplifications we use here, the state space explored is huge. Modelling a robot's position on an $n \times n$ grid, with 4 directions, and two motion modes for r robots requires of the order of $(n \times n \times 4 \times 2)^r$ states to be explored. The state space increases further with variables to deal with turn taking etc. For example, in the case of three robots, using non-strict turn taking within a grid of 7×7, we have more than a thousand million states. Because the underlying model used by the model checker is the product of each robot transition system we will not be able to scale this up to larger number of robots and grid sizes. To combat this we will have to apply and develop some of the work that has been carried out in the model checking field using more sophisticated abstractions, clever representations, and reduction techniques such as slicing or symmetry, in order to limit the state space.

The model checking approach we use here provides the swarm algorithm designer with a systematic way of analysing robot swarms. Whilst we are limited by the state explosion problem, the failing traces obtained in this analysis provide an invaluable starting point for further investigation either to reject as not possible due to the abstraction used or for further consideration by swarm designers to guide simulation or experimentation. Formal verification is a useful tool for deeper analysis of swarm algorithms, and finds more potential faults than simulation or real-robot tests.

Related Work. The application of temporal logics to robots swarms has not been widely used. In [23] temporal logic is used to provide a high level specification of robot swarms however there the focus is on specification rather than verification. We have applied probabilistic model checking to foraging robots, using a counting abstraction, in [14]. In that paper we focus on the mode the robot is in without dealing with either their location or connectedness. In [13] a model checking

approach is adopted to considering the motion of robot swarms. A hierarchical framework is suggested to abstract away from the many details of the problem including the location of the individual robots. Further, the paper assumes a centralised communication architecture which is not assumed here. A related paper is [9] which again considers model checking robot motion but doesn't discuss robot swarms.

7 Conclusions and Future Work

In this paper we have shown how formal verification can be used as part of the development of reliable robot swarm algorithms. Although our examples involving two or three robots might be considered too small to constitute a swarm, they do represent a valid test-case for self-organised flocking or aggregation algorithms, such as the case study of this paper. Indeed swarms of two or three robots are useful in that they test the lower limit bounds of swarm size and are therefore more demanding of the algorithm than larger swarm sizes with higher values of α, as tested in simulation studies. Further the production of failing traces provides a focus for further consideration by the swarm designer.

We clearly need further work to tackle the state explosion problem. This will involve development of abstractions and reduction techniques relevant to swarm algorithms as well as the application of techniques such as symmetry detection to allow the analysis of larger grids and swarms sizes. Additionally we could use probabilities to represent uncertainty such as the unreliability of the robot sensor near to the maximum range leading to the use of probabilistic model checkers such as PRISM [12]. Future work involves not only further analysis of the alpha algorithm but also studying more complex swarm algorithms such as Nembrini's Beta algorithm [17] and the Omega-algorithm developed by Bjerknes [4]. Other properties apart from connectedness such as emergent swarm taxis toward a beacon (i.e. an Infra-Red light source) can also be considered.

References

1. Alur, R., Dang, T., Ivančić, F.: Counterexample-Guided Predicate Abstraction of Hybrid Systems. Theoretical Computer Science 354(2), 250–271 (2006)
2. Arai, T., Pagello, E., Parker, L.: Editorial: Advances in Multi-Robot Systems. IEEE Trans. Robotics and Automation 18(5), 655–661 (2002)
3. Beni, G.: From Swarm Intelligence to Swarm Robotics. In: Şahin, E., Spears, W.M. (eds.) Swarm Robotics 2004. LNCS, vol. 3342, pp. 1–9. Springer, Heidelberg (2005)
4. Bjerknes, J.D.: Scaling and Fault Tolerance in Self-organised Swarms of Mobile Robots. Ph.D. thesis, University of the West of England (2010)
5. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, p. 359. Springer, Heidelberg (2002)
6. Clarke, E., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)

7. Clarke, E.M., Grumberg, O.: Avoiding The State Explosion Problem in Temporal Logic Model Checking. In: ACM Symposium on Principles of Distributed Computing (PODC), pp. 294–303 (1987)
8. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. J. ACM 50(5), 752–794 (2003)
9. Fainekos, G., Kress-Gazit, H., Pappas, G.: Temporal Logic Motion Planning for Mobile Robots. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2020–2025 (2005)
10. Hamann, H.: Space-Time Continuous Models of Swarm Robotic Systems: Supporting Global-to-Local Programming. Cognitive Systems Monographs. Springer, Heidelberg (2010)
11. Henzinger, T., Ho, P.H., Wong-Toi, H.: HYTECH: A Model Checker for Hybrid Systems. International Journal on Software Tools for Technology Transfer 1(1-2), 110–122 (1997)
12. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
13. Kloetzer, M., Belta, C.: Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions. IEEE Transactions on Robotics 23, 320–330 (2007)
14. Konur, S., Dixon, C., Fisher, M.: Formal Verification of Probabilistic Swarm Behaviours. In: Dorigo, M., Birattari, M., Di Caro, G.A., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L.M., Groß, R., Şahin, E., Sayama, H., Stützle, T. (eds.) ANTS 2010. LNCS, vol. 6234, pp. 440–447. Springer, Heidelberg (2010)
15. Martinoli, A., Easton, K., Agassounon, W.: Modeling Swarm Robotic Systems: A Case Study in Collaborative Distributed Manipulation. International Journal of Robotics Research 23(4), 415–436 (2004)
16. Milutinovic, D., Lima, P.: Cells and Robots: Modeling and Control of Large-Size Agent Populations. Tracts in Advanced Robotics, vol. 32. Springer, Heidelberg (2007)
17. Nembrini, J.: Minimalist Coherent Swarming of Wireless Networked Autonomous Mobile Robots. Ph.D. thesis, University of the West of England (2005)
18. Sahin, E., Winfield, A.F.T.: Special issue on Swarm Robotics. Swarm Intelligence 2(2-4), 69–72 (2008)
19. Spears, W.M., Spears, D.F., Hamann, J.C., Heil, R.: Distributed, Physics-Based Control of Swarms of Vehicles. Autonomous Robots 17(2-3), 137–162 (2004)
20. Støy, K.: Using situated communication in distributed autonomous mobile robotics. In: SCAI 2001: Proceedings of the Seventh Scandinavian Conference on Artificial Intelligence, pp. 44–52. IOS Press, Amsterdam (2001)
21. Truszkowski, W., Hallock, H., Rouff, C., Karlin, J., Rash, J., Hinchey, M., Sterritt, R.: Swarms in Space Missions. In: Autonomous and Autonomic Systems: With Applications to NASA Intelligent Spacecraft Operations and Exploration Systems. NASA Monographs in Systems and Software Eng., pp. 207–221. Springer, Heidelberg (2009)
22. Winfield, A., Liu, W., Nembrini, J., Martinoli, A.: Modelling a wireless connected swarm of mobile robots. Swarm Intelligence 2(2-4), 241–266 (2008)
23. Winfield, A., Sa, J., Fernández-Gago, M.C., Dixon, C., Fisher, M.: On Formal Specification of Emergent Behaviours in Swarm Robotic Systems. International Journal of Advanced Robotic Systems 2(4), 363–370 (2005)
24. Yuh, J.: Design and Control of Autonomous Underwater Robots: A Survey. Autonomous Robots 8(1), 7–24 (2000)