ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/224563797

From Fireflies to Fault-Tolerant Swarms of Robots

Article *in* IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council · September 2009

DOI: 10.1109/TEVC.2009.2017516 · Source: IEEE Xplore

CITATIONS
141

READS
572

3 authors:

Anders Lyhne Christensen
University of Southern Denmark
161 PUBLICATIONS **2,513** CITATIONS

SEE PROFILE

Rehan O'Grady
Université Libre de Bruxelles
62 PUBLICATIONS **1,773** CITATIONS

SEE PROFILE

Marco Dorigo
Université Libre de Bruxelles
523 PUBLICATIONS **95,018** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Evolutionary Online Behaviour Learning and Adaptation in Real Robots View project

Project Evolution of self-organization View project

All content following this page was uploaded by Anders Lyhne Christensen on 19 May 2014.

The user has requested enhancement of the downloaded file.

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

From Fireflies to Fault Tolerant Swarms of Robots

Anders Lyhne Christensen, Rehan O'Grady
and Marco Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2008-002

January 2008

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2008-002

Revision history:
TR/IRIDIA/2008-002 January 2008

The information provided is the sole responsibility of the authors and does not necessarily reflect the opinion of the members of IRIDIA. The authors take full responsability for any copyright breaches that may result from publication of this paper in the IRIDIA – Technical Report Series. IRIDIA is not responsible for any use that might be made of data appearing in this publication.

From Fireflies to Fault Tolerant Swarms of Robots

Anders Lyhne Christensen and Rehan O'Grady and Marco Dorigo

IRIDIA, CoDE, Université Libre de Bruxelles,
50, Av. Franklin. Roosevelt CP 194/6,
1050 Bruxelles, Belgium
alyhne@iridia.ulb.ac.be,{rogrady,mdorigo}@ulb.ac.be

Abstract. One of the essential benefits of multi-robot systems is redundancy. In case one robot breaks down, another robot can take steps to repair the failed robot or take over the failed robot's task. Although fault tolerance and robustness to individual failures have often been central arguments in favor of multi-robot systems, few studies have been dedicated to the subject. In this study, we take inspiration from the synchronized flashing behavior observed in some species of fireflies. We derive a completely distributed algorithm to detect non-operational individuals in a multi-robot system. Each robot flashes by lighting up its on-board LEDs and neighboring robots are driven to flash in synchrony. Since robots that are suffering catastrophic failures do not flash periodically, they can be detected by operational robots. We explore the performance of the proposed algorithm both on a real world multi-robot system and in simulation. We show that failed robots are detected correctly and in a timely manner, and we show that a system composed of robots with simulated self-repair capabilities can survive relatively high failure rates.

1 Introduction

Nature has produced a multitude of remarkably robust and adaptive systems. These qualities are often derived from underlying self-organization mechanisms and from massive built-in redundancy. Examples can be found at scales from the nano to the macro: nanostructures [1], the architecture of a cell [2], ensembles of cells forming organs such as a human heart [3], and societies of insects [4]. As engineers, we can learn from and try to imitate such natural systems in which complex behaviour results from the basic rules of interaction between essentially simple components.

In this study, we leverage some of the high-level principles behind synchronizing systems found in Nature to obtain a robust, simple, distributed approach to fault detection in groups or swarms of autonomous robots. By detecting faults, the robots can leverage their multiplicity and ensure continued operation by reassigning functional robots to the failed robots' task or by taking steps to have the failed robots repaired. In a recent paper [5], the reliability of seven mobile robots from three different manufacturers was tracked over a period of two years and the average mean time between failures was found to be 8 hours. The result suggests that faults in mobile robots are quite frequent. As the number of constituent robots increases, we would expect the rate of failure to grow correspondingly. Faults are therefore likely to be common events in multi-robot systems.

Many studies have been devoted to *endogenous fault detection*, that is, a robot detecting faults in itself, see for instance [6, 7, 8, 9, 10, 11, 12, 13, 14]. Some faults are, however, hard to detect in the robot in which they occur. These faults include software bugs that cause a control program to hang, sensor failures that prevent a robot from detecting that something is wrong, and mechanical faults such as an unstable connection to a power source. Alternatively, a robot might be able to detect a fault, but the fault itself might still render the robot unable to alert other robots or a human operator. The robustness of a multi-robot system can therefore be improved by giving robots the ability to detect faults in one another. We refer to this process as *exogenous fault detection*.

Exogenous fault detection and fault tolerance in multi-robot systems have been studied in [15, 16, 17, 18, 19]. In the majority of the proposed approaches, however, the robots are required to be tightly coupled and/or radio communication is used to facilitate fault detection. One of the most

widely cited approaches for achieving fault tolerance in multi-robot systems, ALLIANCE [15], requires that the robots are able to track the progress of several tasks at the same time. Although ALLIANCE does not require explicit communication between robots, the method has been shown to operate better if such communication means are used. As the number of robots grows, tightly coupled systems become harder to realize due to scalability issues.

When we look to Nature, we seldom find centralized approaches. Instead, we find numerous examples of decentralized and self-organized systems: schools of fish changing direction at the same time and never colliding, trail formation in ants, and termite mound construction without a pre-defined blueprint or a central coordination mechanism [20]. The philosophy behind Nature-inspired swarm robotics is to rely on self-organization through local interactions between robots. The potential advantages of designs adhering to this philosophy include scalability, inherent parallelism, and robustness to individual failures [21]. The approach that we advocate in this study is completely distributed. Through local interactions, a group of robots is able to synchronize and reach a state in which they flash periodically in unison. When a robot breaks down it also ceases to flash. By detecting the absence of flashes, operational robots can effectively detect failed robots.

This paper is organized as follows: In Sect. 2 we discuss previous studies related to synchronization among pulse-coupled oscillators and studies related to fault detection in robots. In Sect. 3 we present the robotic hardware (the *swarm-bot* platform) used in this study. In Sect. 4 we show how simulated and real robots can synchronize their flashing and we explore various parameters such as robot density, group size and coupling strength. In Sect. 5 we show how non-operational robots can be detected and we provide results for a system where the robots have (simulated) self-repair capabilities. Concluding remarks and directions for future research are provided in Sect. 6.

2 Related Work

2.1 Synchronization in Natural and Artificial Systems

In Nature we find many examples of coupled oscillating systems that lead to various types of synchronous behavior. The canonical example is large groups of tropical fireflies, found on river banks in Southeast Asia, that spontaneously synchronize their rhythmic flashes [22, 23]. Other examples include cardiac cells [24], choruses of grasshoppers [25], female menstrual cycles [26], and clapping in theaters [27].

Systems of coupled oscillators can be divided into two classes: oscillators that continuously influence one another (see for instance [28]) and so called *integrate-and-fire* or *pulse-coupled* oscillators, where one oscillator only influences other oscillators during short, periodic pulses. In this study, we focus on the latter type. The internal state or *activation* of each oscillator increases over time until it reaches a certain threshold. When the threshold is reached, the oscillator discharges (*fires*) and the activation instantly jumps back to zero – the cycle then repeats. When a nearby oscillator observes a flash it immediately increases its activation by a (small) amount. If this increase causes the oscillator's activation to exceed the firing threshold, the oscillator fires, resets its activation to zero, and commences a new cycle. Analytically, many pulse-coupled networks can be written in the following form [29]:

$$\dot{x}_i = f(x_i) + \epsilon \sum_{j \in \mathsf{N}} h\big(x_i\big)\delta(t - t_j^*), \tag{1}$$

where $x_i \in [0, 1]$ denotes the activation of oscillator i. The function f describes its dynamics. The constant ϵ defines the strength of the coupling between oscillators. N is the set of oscillator i's neighbors. The pulse-coupling function h describes the effect of the firing of another oscillator j on i. The time t_j^* marks the moment when j last fired. The delta function $\delta(t)$ satisfies that $\delta = 0$ for all $t \neq 0$, $\delta(0) = \infty$ and $\int \delta = 1$. An example with two oscillators for which f is constant and h is linear is shown in Fig. 1.

The self-synchronization of pulse-coupled oscillating cardiac pacemaker cells was first described by Peskin [3]. Mirollo and Strogatz later showed that a population of fully connected pulse-coupled

Fig. 1: An example of two pulse-coupled oscillators. Both oscillators increase at a constant rate until the threshold is reached – at which point the oscillator fires and jumps back to 0 – or until the firing of the other oscillator is observed. When one oscillator observes the other's firing, it increases its own state by $\epsilon\, h(x)$, where ϵ is the pulse-coupling constant, $h(x)$ the pulse-coupling function, and x the activation of the oscillator.

oscillators almost always evolves to a state in which all oscillators are firing synchronously [30]. Recently, Lucarelli and Wang [31] showed that a group of pulse-coupled oscillators will eventually synchronize even when each oscillator interacts with only a subset of the population. This holds true for systems with changing topologies as long as the interaction graphs are connected.[1]

Understanding synchronization is not only important for describing natural phenomena – synchronization is a central issue in distributed computing and distributed sensing. The problem of establishing a consistent global time base across nodes in a distributed system subject to message delays, network congestion, node failures, and clock skews has received a great deal of attention (see for instance [32, 33]). The behavior of fireflies has inspired algorithms for heartbeat synchronization in overlay networks [34], imposing reference timing in wireless networks [35], and in sensor networks for coordinating sensing and communication [36].

In this study, we rely on local visual communication. We are therefore not faced with issues such as variable propagation delays and congestion that several studies on synchronization across data networks have had to deal with.

2.2 Fault Detection and Fault Tolerance in Multi-Robot and Swarm Robotic Systems

Parker [15] has demonstrated that cooperating teams of robots based on the ALLIANCE software architecture can achieve a high degree of fault tolerance. Fault tolerance is obtained by modelling "motivations" mathematically and by adaptive task selection based on these motivations. When a robot fails to register satisfactory progress in its current task (for instance due to the presence of a fault), it decreases its motivation for performing the task. Eventually, the robot will switch to another task that it may still be able to perform. Alternatively, another robot will discover that there is limited or no progress in the task undertaken by the failed robot, and take over. Other approaches such as MURDOCH [16, 17] and TraderBots [18] have been proposed. In both MURDOCH and TraderBots explicit communication is used to negotiate task allocation. Fault detection and fault tolerance are built into this negotiation process.

In [37], Li and Parker use sensor analysis to facilitate fault detection in a tightly-coupled multi-robot team. Their "sensor analysis for fault detection" (SAFDetection) is based on data collection during normal operation and subsequent abnormality detection. The approach combines data clustering techniques with the generation of a probabilistic state diagram to model normal operation of the multi-robot system. The whole multi-robot team is regarded as a single monolithic entity with a unified set of sensors. This puts limits on the size of the teams since the amount of data communicated and processed centrally is proportional to the number of robots in the team. In [37], the authors state that they intend to extend their approach to distributed monitoring.

In some cases, fault tolerance is an inherent property of the system and is not handled explicitly. Lewis and Tan [19] have shown that their control algorithm for maintaining geometric formations exhibits correct behavior even if one of the robots fails. Winfield and Nembrini [38] performed failure mode and effect analysis (FMEA) on a containment task for a swarm of robots connected through local wireless links. The authors found that their system exhibited a high level of tolerance to individual failures, but at the same time that certain types of faults could effectively anchor the swarm at the failed robot's position. In their conclusion, they envisage a new behavior in which the swarm can identify failed members and isolate them from the rest of the swarm. In both [19] and [38] mentioned above, fault tolerance is a consequence of the simple and adaptive nature of the controller design and not of explicit fault detection, diagnosis and accommodation. Although attempts to generalize fault tolerance by design have been made (see for instance [39]), it is unknown whether such designs are feasible (or even theoretically possible) for all systems and all tasks.

In this paper, we study a completely distributed approach that builds on the principles behind synchronization observed in fireflies to implement a heartbeat-like fault detection scheme for groups

[1] We obtain the interaction graph for a population of oscillators by letting every oscillator correspond to a node in the graph with an edge to each member of its neighbor set.

of robots. Fault tolerance is achieved by explicit exogenous fault detection (as opposed to implicit fault tolerance by design). Unlike many previously studied approaches, the approach that we propose does not depend on radio communication, a centralized coordination mechanism, or a strict definition of tasks and progress.

3 Hardware Platform

For our experiments, we use the *swarm-bot* robotic platform [40]. This innovative platform was designed and built by Francesco Mondada's group at the Laboratoire de Systme Robotiques (LSRO) of EPFL. The system consists of a number of mobile autonomous robots called *s-bots* (see Fig. 2) that are capable of forming physical connections with each other. The *swarm-bot* platform has been used for several studies, mainly in swarm intelligence and collective robotics (see for instance [41, 42]). Overcoming steep hills and transport of heavy objects are notable examples of tasks which a single robot could not complete individually, but which have been solved successfully by teams of collaborating robots [43, 44, 45].

Each *s-bot* is equipped with an XScale CPU running at 400 MHz, a number of sensors including an omni-directional camera, light and proximity sensors. Physical connections between *s-bots* are established by a gripper-based connection mechanism. Each *s-bot* is surrounded by a transparent ring that can be grasped by other *s-bots*. *S-bots* can advertise their location by means of eight sets of RGB-colored LEDs distributed around the inside of their transparent ring (see Fig. 3).

Fig. 2: *S-bot*: an autonomous, mobile robot capable of forming physical connections with other *s-bots*.

Control programs for *s-bots* operate in a discrete manner: A control program is run as a succession of sense-think-act cycles. In each cycle, the control program reads data from sensors such as the on-board camera, infrared proximity sensors, and so on, processes the data, and sends control signals to the actuators such as the motors that drive the robot. In this study, we use a control cycle period of 0.15 s.

Fig. 3: A: an *s-bot* with all its LEDs off. B: an *s-bot* with its red LEDs illuminated.

3.1 The *S-bot* Camera and Image Processing

The *s-bots* are equipped with an omni-directional camera mounted in the turret. The camera points upwards at a spherical mirror mounted in a transparent perspex tube above the turret. The *s-bot* camera captures images of the robot's surroundings reflected in the semi-spherical mirror. The camera sensor records 640x480 color images. The *s-bots* have sufficient on-board processing power to scan entire images and identify objects based on color. Images are divided into a grid of multi-pixel blocks and the image processor outputs the prevalent color in each block (or indicates the absence of any relevant color). We have configured the image processor to detect the location of the colored LEDs of the *s-bots* and discard any other information. Depending on light conditions, an *s-bot* can detect LEDs on neighboring *s-bots* up to 50 cm away. In order for nearby robots to be able to always see one another, robots illuminate their green LEDs when they are not flashing.

3.2 Simulation Environment

We use a custom software simulator [46]. All the sensors and actuators shown in Fig. 2 and in Fig. 3 have been implemented in the simulator. Combined with an implementation of the robot's application programming interface (API), this allows programs developed for the real *s-bots* to be run in the simulator and vice versa. Although differences exist between the real world and simulation, a software simulator allows us to experiment with setups that, for instance, contain more robots than we have access to in reality. Also, in a simulator, we can easily conduct a large number of replications of an experiment and thereby discover trends in the behavior of the system.

4 Synchronization in Robots

We propose an approach for synchronization based on local visual communication. The approach resembles behavior observed in fireflies: we let each robot act as an integrate-and-fire oscillator and when the activation of the oscillator reaches a certain threshold, the robot lights up its red LEDs like in the example shown in Fig. 3 and resets its oscillator. When neighboring robots (within 50 cm) detect the flash, they increment their own activation.

Visual communication has the advantage over radio communication that we do not need to be concerned with establishing and maintaining radio links. When a robot fires by lighting up its red LEDs, the flash is automatically "broadcast" to all robots within visual range.

4.1 Discrete Oscillators

Due to the inherent discreteness of the sense-think-act control paradigm, we transform the continuous model in Eqn. 1 into a discrete model with linear dynamics:

$$x_i(n+1) = x_i(n) + \frac{1}{T} + \epsilon\, \alpha_i(n)\, h\big(x_i(n)\big), \tag{2}$$

where $x_i(n)$ is the activation of robot i at control cycle n. T is the period between flashes of an isolated robot. In this study, we have chosen T to be 100 control cycles (corresponding to 15 s). We experiment with different values for the pulse-coupling constant ϵ in Sect. 4.4.2. $\alpha_i(n)$ is the number of flashing robots seen by i at control step n. We use the linear pulse-coupling function:

$$h(x) = x \tag{3}$$

When $x_i(n)$ exceeds 1, robot i flashes and its activation is reset to 0. There is a (small) latency between the moment that the control program sends a signal to the flash LEDs and until the moment they respond. Before a neighboring robot can perceive a flash, it must have already recorded and processed a frame from its on-board camera in which the flash is visible. This step entails an additional latency. Furthermore, images from the camera are retrieved and processed asynchronously by the on-board software in a separate execution thread. We have experimentally found that we can compensate for these delays by keeping the flash LEDs on for 5 control cycles (0.75 s). Flash spans of this length allow the robots to reach and remain in a synchronized state. When robots in a synchronized system keep their flash LEDs on for 5 consecutive control cycles, they perceive the flashes from other robots, while they are flashing themselves. This creates a stable fix-point for the system. We compute $\alpha_i(n)$ based on the most recent frame recorded by the on-board camera. In order to prevent the robots from perceiving the same flashes multiple times, we compute $\alpha_i(n)$ by counting new flashes only, that is, the flashes that were not perceived in the previous control cycle.

4.2 Synchronization Experiments in Simulation

We are interested in the time it takes for all robots to synchronize. We define the system to be synchronized when it is in a state where the value of every activation value $x_i(n)$ is no further than $1/T$ from all other activations. An example of the development of activation values sampled every T during a run with 25 robots in simulation is shown in Fig. 4. The activation for each robot at every T–th simulation step is plotted as a cross. In the example, the robots synchronize after 435 s.

The time it takes for a swarm of robots to synchronize depends, apart from the parameters ϵ and T, also on the density of robots, on how the robots move, and on the total number of robots. In a given environment, the density of robots together with the total number of robots define the average degree and the diameter of the interaction graph, while the pattern of movement for the robots defines its dynamic properties. As the pattern of movement is strongly task-dependent, we limit our experiments to two extreme cases: one in which all robots perform a random walk and one in which all robots are static. In both types of experiments the robots start at random positions and with random orientations. In the experiments with static robots a check is performed before the start of each experiment to ensure that the interaction graph is connected. In case it is not, all the robots are repositioned until a configuration is found that produces a connected interaction graph.

In simulation we have explored the rate of synchronization for swarms of 10, 25, 50, and 100 robots (see Fig. 5), at densities of 2, 4, 6, 8, and 10 robots/m^2 in square-shaped arenas (see Fig. 6), and for coupling strengths ϵ ranging from 0.01 to 0.50 (see Fig. 7). In all figures, one bar represents the mean synchronization rate observed in 100 replications of the experiment and the error-bars denote the standard deviation.

For all experimental configurations, moving robots tend to synchronize faster than static robots. Visual inspection of the experiments confirmed that a system of static robots in many cases reaches a state of near-synchrony, where flashes propagate in waves through the swarm before the robots synchronize. Snapshots of a flash wave propagating through a static swarm of robots is shown in Fig. 8. When the flash wave starts, all the robots have activations close to the firing threshold

Fig. 4: An example of the evolution of activations sampled every T in 25 mobile robots over the course of 10 minutes. One cross represents the activation for a single robot at the corresponding time.

Fig. 5: Synchronization rate in groups of 10 to 100 simulated robots. Each bar summarizes 100 runs and error-bars denote the standard deviation. The density was 8 robots/m^2 and a coupling constant of $\epsilon = 0.1$ was used in all runs.

Fig. 6: Synchronization rate in a group of 50 simulated robots at different densities. Each bar summarizes 100 runs and error-bars denote the standard deviation. A coupling constant of $\epsilon = 0.1$ was used in all runs. Due to the limited sensory range of the *s-bots* (up to 50 cm) experiments with static robots at a density of 2 robots/m^2 were not conducted. At this density, the interaction graph is almost never connected when the robots are distributed randomly.

Fig. 7: Synchronization rate a group of 50 simulated robots for coupling constants $0.01, 0.02, 0.05, 0.10, 0.20, 0.50$. Each bar summarizes 100 runs and error-bars denote the standard deviation. The density was 6 robots/m^2.

and they therefore flash as soon as a nearby robot flashes. Flashes would not propagate in waves if robots could perceive and respond to flashes instantly, because as soon as one robot flashes, all robots would flash (and they would be synchronized). Wave propagation of flashes can thus occur due to the latencies associated with turning on the flash LEDs in the flashing robot and the image capture and image processing.

Fig. 8: An example of a flash wave in a group of static robots.

In moving robots, the wave propagation phenomenon is not as pronounced. Robots that are close to each other have similar activation values when flashes propagate in waves. However, when the robots move, the interaction graph changes. This means that the individual robots do not remain at the same distance from the origin of the flash wave as the system evolves. When individuals that are close to the robot that triggers the flash waves move away, they cause other robots (more distant from the wave origin) to flash sooner. Similarly, as individuals further away from the robot that triggers the flash wave move closer to the wave origin, they are driven to flash sooner, thus speeding up the global synchronization process.

The synchronization rate scales linearly (with a gentle slope) with swarm sizes up to 100 *s-bots* (see Fig. 5). The mean synchronization rate for a group of 10 *s-bots* is 62 s, while for 100 *s-bots* the rate is 164 s – less than 3 times as long.

The synchronization rate at different densities plotted in Fig. 6 shows that denser swarms tend to synchronize faster. When a swarm is dense, more members are within each others' sensory range. The results indicate that the larger the subset of robots each individual interacts with, the faster the overall group synchronizes.

The strength of each interaction is controlled by the coupling constant ϵ. The results in Fig. 7 show that if ϵ is large a swarm tends to synchronize faster. Setting ϵ too high is, however, problematic when we want to detect faults because one robot – including a failed one – has a significant effect on its neighbors. In this case, a robot that experiences a catastrophic fault while its flash LEDs are illuminated can effectively disrupt the whole system. Furthermore, for large swarm sizes, high values of ϵ can make the system unstable and prevent it from synchronizing.

4.3 Synchronization Experiments with Real Robots

We conducted two sets of experiments with 10 real robots: one set of experiments with static robots and one set of experiments with moving robots (random walk and obstacle avoidance). The experiments were performed in a walled arena with dimensions 1.6 m x 1.6 m (yielding a density of 4 robots/m^2). The coupling constant ϵ was set to 0.1 and the flash period T was 100 control cycles. The robots were assigned different initial random activations. The initial positions for the robots were obtained in the same way as in simulation (see Sect. 4.4.2). Based on video recordings, the synchronization rate was measured as the time from the frame in which the robots were started until the first frame in which all the robots had their flash LEDs illuminated. Fig. 9 shows an example of synchronized robots flashing. The experimental setups with static robots and with moving robots, respectively, were replicated 10 times with different initial conditions. A summary of the results is shown in Tab. 1. Videos of the experiments can be downloaded from [47].

Table 1: Synchronization rate for 10 real robot.

	Mean	St.dev.	Shortest	Longest
Static	94 s	72 s	55 s	174 s
Moving	77 s	28 s	30 s	118 s

Fig. 9: A photo of synchronized robots flashing at the same time.

In real robots we observe the same trend as in simulation: moving robots tend to synchronize faster than static robots. The mean synchronization rate of static robots was 94 s while the mean synchronization rate for mobile robots was 77 s. In all 10 experiments with static robots and in all 10 experiments with moving robots, the robots synchronized. The results indicate that real robots operating as pulse-coupled oscillators are able to synchronize despite the discrete nature of the control sense-think-act paradigm and despite the inherent latencies associated with the sensory and actuator systems.

5 Fault Detection

Synchronization can be used as an exogenous fault detection tool if the robots assume that a robot that is not flashing has a fault. A robot can stop flashing voluntarily if it detects a fault in itself. In this way, it can implicitly signal that it requires assistance. A robot also stops flashing

when it experiences a catastrophic fault (software bug, physical damage, and so on...) which causes the control program and thus the periodic flashing to stop. When operational robots discover a non-flashing teammate they know that a fault has occurred and they can take steps to rectify the situation. Conceptually, the scheme is straightforward. However, two issues need to be addressed in order for the scheme to be implemented on real robots: it cannot be assumed that the robots are always synchronized and the sensory range of the robots is limited.

5.1 Detecting Faults in Non-Synchronized Robots

In a normal situation the robots would be operational and synchronized (see Fig. 10a). However, when robots commence a task or when they encounter each other after having been separated for a period of time, their activations are likely to differ. In other words, they are not synchronized. This means that one robot cannot assume that another robot has become non-operational just because the two robots do not flash in unison. To address this issue, a flashing robot does not immediately consider another robot non-operational if the two robots do not flash at the same time. Instead, the flashing robot (F) treats the robot (N) that did not flash when F flashed as a *candidate* robot. We say that F becomes *suspicious* of N. If N flashes before F flashes again, both robots are operational but they are just not (yet) synchronized (see Fig. 10b). However, if F flashes *again before* N flashes, F assumes that N is non-operational (see Fig. 10c). Hence, a robot detects a fault if it flashes twice while observing that another robot does not flash at all.

There is however a problem with this scheme. In fact, there is a rare situation in which one operational robot (*R2*) can flash twice while another *operational* robot (*R1*) does not flash a single time. This can happen when *R1* flashes right before *R2* and when *R2* subsequently perceives sufficient flashes to increase its activation so much that it flashes again before *R1* flashes a second time. However, R2's second flash will often provoke R1 to flash. R2 can, in fact, calculate the sufficient conditions under which it's second flash will provoke R1 to flash. When these conditions are met and R2's second flash does not provoke R1 into flashing, R2 can safely assume that R1 has a fault. We let Δ denote the amount by which *R2*'s activation has been increased due to flashes from other robots. In the worst case, *R1*'s activation has not been advanced by any flashes. When *R2* reaches the firing threshold ($= 1$) *R1*'s activation is therefore at most Δ away from the firing threshold, i.e. *R1*'s activation is at least $1 - \Delta$. Assuming that the two robots perceive each others' flashes, *R2*'s second flash will increase *R1*'s activation by at least $\epsilon h(1 - \Delta)$. Thus, *R2*'s second flash will drive *R1* to flash if:

$$\epsilon h(1 - \Delta) \geq \Delta. \tag{4}$$

Thus if R2 flashes twice while R1 does not flash at all (including not being provoked to flash by R2's second flash) and if Eqn. 4 is true, R2 can conclude that R1 has a fault. Otherwise R2 must wait until its next flash to determine whether or not R1 is operational. If R1 has still not flashed in the meantime it must have a fault.

The case in which a robot breaks down *while* it is flashing is not caught by the scheme presented above. In other words, no robot would ever become suspicious of a robot that becomes non-operational while its flash LEDs are illuminated. Consequently, the non-operational robot would never be detected. Faults that occur while the robot is flashing, leaving the flash LEDs illuminated, however, can easily be detected: when a robot's activation passes its midpoint (0.5), it becomes suspicious of any robot that has its flash LEDs illuminated. If the candidate robot still has it LEDs on after the normal flash span (5 control cycles), the suspicious robot can conclude that the candidate robot with the flash LEDs on is not operating correctly. This situation is illustrated in Fig. 10d.

5.2 Limited Sensory Range

Since the robots are mobile and since their sensors have a limited range, the robots can come into and exit each other's view repeatedly. We do not assume that robots have unique IDs or that they

a) Both robots operational and synchronized:

b) Both robots operational but not synchronized:

c) One robot failed:

d) One robot failed with flash LEDs on:

Legend:

can identify each other. In other words, a robot cannot keep track of the flashing activity of every other robot in the swarm. Therefore, whenever a robot becomes suspicious it stops in order to keep the candidate robot within sensory range until it can determine if the candidate robot has a fault or not.[2]

5.3 Time Overhead

When a group of robots start a new task, they are not always synchronized. This means that the robots do not flash at the same time. While a group of robots is in the process of synchronizing, they frequently become suspicious. While they are suspicious, they stop performing the task and wait while they determine if the candidate robot is non-operational or if it is just not synchronized. This has a negative impact on the performance of the group as time that could have been used for carrying out a task is spent on being suspicious. In Fig. 11 we have plotted the average percentage of control cycles that the robots were suspicious in the beginning of an example run.

Fig. 11: Average percentage of the control cycles spent in the suspicious state over intervals of 15 s during a run with 50 simulated robots in a 2.5 m x 2.5 m arena. The robots were not initially synchronized.

The time initially spent by the robots on being unnecessarily suspicious while a group of robots is synchronizing can be reduced or eliminated entirely by introducing a *warm-up period*. During the warm-up period a robot ignores any indications of faults and does not become suspicious. If we had introduced a warm-up period of 120 s or longer in the experiment summarized in Fig. 11, none of the robots would have become suspicious during the initial synchronization period, and the initial overhead of the stop-while-suspicious strategy would have been eliminated. However, there is a trade-off between the length of the warm-up period and the latency of fault detection since faults cannot be detected during the warm-up period.

5.4 Implementation

An overview of the control and fault detection logic executed every control cycle is shown in Alg. 1. The activation x is incremented by the sum of the constant increase $1/T$ and the product of the

[2] For simplicity we assume that a catastrophic fault causes a robot to stop its movement. If this were not the case, then a suspicious robots would have to follow the candidate robot and stay within visual range until it could determine whether or not the candidate robot has a fault.

coupling strength ϵ, the number of flashes seen α, and the pulse-coupling function $h(x)$. When x surpasses 1, the robot flashes and checks for non-flashing robots, while if x has just passed 0.5, a check is made to determine if any neighboring robots has become non-operational with its flash LEDs illuminated. In case a candidate robot is found, the robot stops and waits until it can be concluded whether the candidate is operational or not. If no candidate was found, the robot performs a random walk while avoiding obstacles.

The logic for checking for non-flashing candidates and for candidates with their flash LEDs illuminated is shown in Alg. 2 and Alg. 3, respectively.

Algorithm 1: ControlCycle()

```
ReadSensors();
if HasFlashed(candidate) then
    candidate = none;
end
x ← x + 1/T + εαh(x);
Δ ← Δ + εαh(x) ;
if x > 1 then
    FlashAndCheckForFailedRobots();
    x ← 0;
end
if x has passed 0.5 then
    CheckForFailedRobotsWithFlashOn();
end
if candidate ≠ none then
    StopMoving();
else
    RandomWalkAndAvoidObstacles();
end
```

Algorithm 2: FlashAndCheckForFailedRobots()

```
TurnOnFlashLeds(5 cycles) ;
if candidate = none then
    candidate = CheckForNonFlashingRobots();
    Δ = 0;
else
    if εh(1 − Δ) ≥ Δ then
        failedrobot ← candidate;
        ...A fault has been detected. Take
        ...actions to deal with the fault.
    else
        ...Wait until next flash before concluding
        ...if the candidate robot has failed or not.
    end
end
```

Algorithm 3: CheckForFailedRobotsWithFlashOn()

```
candidate = CheckForFlashingRobots();
if candidate ≠ none then
    StopMoving();
    if CandidateRobotStillFlashing() after 5 control cycles then
        failedrobot ← candidate;
        ...A fault has been detected. Take
        ...actions to deal with the fault.
    end
end
```

5.5 Fault Detection Experiments with Real Robots

In 10 experiments we measured the time it took for one or more robots to detect and react to a failed robot. We took the first steps towards a scenario in which one or more robots can facilitate the repair of a failed robot – either directly or by physically connecting and transporting the failed robot[3] to a special zone where the robot is then repaired or replaced. The experiments were performed in the same arena and with the same parameter settings as the syncronization experiments for mobile robots described in Sect. 4.4.3 (an arena of 1.6 m x 1.6 m, $\epsilon = 0.1$, and $T = 100$ control cycles). In each experiment, we let a group of 10 robots synchronize and then we simulated a catastrophic fault in one of the robots. We measured the time from the moment a fault was injected until one of the operational robots reacted to the fault by detecting the fault and physically connecting to the failed robot. The results are shown in Tab. 2.

The fault was correctly detected in all 10 experiments. The mean reaction time was 53.2 s. This result includes the times required for the following activities: an operational robot detects the absence of a flash, the operational robot is suspicious for up to T (15 s), the operational robot navigates to and grasps the failed robot.

The shortest reaction time to a fault was 30.2 s. In the corresponding experiment, the fault was injected just before the other robots in the swarm flashed and a nearby robot therefore became suspicious less than a second after the fault had been injected. Furthermore, the robot that detected the fault was close to the failed robot and had an orientation that allowed it to quickly connect to the failed robot. In the experiment in which the longest reaction time (135.5 s) was observed, at first only a single operational robot detected the fault. It unsuccessfully attempted to grasp the failed robot twice. Eventually another operational robot detected the fault and connected to the failed robot.

Table 2: Fault reaction time results on real robots

Mean reaction time	53.2 s
Standard deviation	31.3 s
Shortest reaction time	30.2 s
Longest reaction time	135.5 s

In one of the experiments a real fault occurred. After an operational robot had detected and connected to the robot in which we had injected a fault, a robot stopped responding due to a hardware I/O error. The error rendered the robot unable to control any of its actuators, including its treels and its LEDs. This real (non-simulated) fault was also detected and an operational robot connected to the failed robot.

5.6 Fault Tolerance Experiments with Real Robots

In order to test our approach in a scenario where more than one robot can become non-operational, we conducted an experiment with a group of 10 robots, in which a fault was injected in an operational robot with a probability of $p = 0.0005$ every control cycle. We simulated a repair mechanism that allowed one robot to "repair" another robot by physically connecting to it and by illuminating its blue LEDs for 15 s. When a failed robot detected that it had been "repaired", it set its activation to a random value and restarted its controller. We let the experiment run for 12 min. All robots were operational from the start of the experiment and the first fault occurred after 20 s. During the experiment a total of 13 simulated faults occurred. At one point a total of

[3] *S-bots* have been shown capable of collectively transporting objects that are larger and heavier than an *s-bot*, see for instance [48].

four robots were non-operational, while only one robot was non-operational when the experiment was stopped.

A robot experienced a real hardware I/O fault similar to the one described above in Sect. 5.5.5. Two neighboring robots detected the fault and connected to the robot with the real fault. After the two robots had connected to the failed robot and performed the repair action, we removed the failed robot from the arena. We let the other nine robots continue while we restarted and reintroduced the failed robot 3 min later. Furthermore, we manually put a robot upright after it had toppled over due to a collision with two other robots. A video of the experiment can be found on [47].

The results suggest that our approach is robust in situations where multiple faults can be present at the same time. Furthermore, when the robots can repair one another, a swarm of robots can survive a relatively high rate of failure.

6 Conclusions and Future Work

In this study, we have presented a distributed approach for detecting non-operational members in swarms of robots. Our algorithm is inspired by the synchronous flashing behavior observed in some species of fireflies. Robots flash periodically by lighting up their on-board LEDs. Whenever a robot perceives a flash from a nearby robot, it increases its own activation and flashes slightly sooner than if it had not seen a flash. We showed that swarms of simulated and real robots following this scheme are driven to flash in synchrony. The rate of synchronization was found to depend on the size of the swarm, the number of robots that each member interacts with, the coupling strength between the robots (the effect of one robot's flash on another nearby robot), and whether the robots move or are stationary.

In our fault detection scheme, the periodic flashes function as a heart-beat mechanism. A failed robot need not explicitly signal other nearby robots that it requires assistance – it only needs to stop flashing. We do not, therefore, need to distinguish between robots that voluntarily have stopped flashing and robots that, for instance, have experienced a catastrophic fault rendering them unable to take any action – including flashing. We showed that real robots are able to detect and respond to faults by detecting non-flashing robots. We also showed that the scheme is robust to multiple faults and that a team of robots with self-repair capabilities is able to survive a relatively high rate of failure.

In many previous studies fault detection was facilitated by global negotiation and radio communication. In contrast, our firefly-inspired fault detection approach is completely distributed and relies on local information only. A potential advantage of a distributed approach is scalability, which becomes an increasingly important factor as larger swarms of, for instance, hundreds or thousands of robots are considered. In our experiments, we found that the rate of synchronization depends on the size of a swarm. This means that as the size of a swarm grows it takes longer for the robots to synchronize. However, swarms need not be *globally* synchronized for our fault detection scheme to work efficiently – it suffices that robots are synchronized *locally* with nearby robots. It would thus be interesting to determine the performance of our approach when a swarm is in a global state of near-synchrony, for instance, when waves of flashes are propagating through the swarm (see Sect. 4. 4.2).

A potential direction for future research is implementing and evaluating the performance of our approach in a real task-execution scenario. While carrying out a task, operational robots could detect and transport failed robots to a pre-designated zone and alert a human operator, who could then repair or replace the failed robots. Another interesting question is how to extend the approach to take advantage of possible heterogeneities in a swarm, e.g. robots with different sensory, manipulation, and/or repair capabilities. This type of heterogeneity could possibly be leveraged to facilitate faster synchronization, faster fault detection and true self-repair, while still allowing for a completely distributed, swarm intelligent approach.

Acknowledgements

This work would not have been possible without the innovative robotic hardware developed by Francesco Mondada's group at the Laboratoire de Systme Robotiques (LSRO) of EPFL. This work was supported by the *SWARMANOID* project, funded by the Future and Emerging Technologies programme (IST-FET) of the European Commission, under grant IST-022888 and by the *VIRTUAL SWARMANOID* project funded by the F.R.S.-FNRS. Anders Christensen acknowledges support from COMP2SYS, a Marie Curie Early Stage Research Training Site funded by the European Community's Sixth Framework Programme (grant MEST-CT-2004-505079). The information provided is the sole responsibility of the authors and does not reflect the European Commission's opinion. The European Commission is not responsible for any use that might be made of data appearing in this publication. Marco Dorigo acknowledges support from the F.R.S.-FNRS, of which he is a Research Director.

References

[1] K. Pohl, M. C. Bartelt, J. de La Figuera, N. C. Bartelt, J. Hrbek, and R. Q. Hwang. Identifying the forces responsible for self-organization of nanostructures at crystal surfaces. *Nature*, 397(6716):238, 1999.

[2] T. Misteli. The concept of self-organization in cellular architecture. *The Journal of Cell Biology*, 155(2):181–186, 2001.

[3] C. S. Peskin. *Mathematical aspects of heart physiology.* Courant Institute of Mathematical Sciences, New York University, New York, 1975.

[4] E. Bonabeau, G. Theraulaz, J. L. Deneubourg, S. Aron, and S. Camazine. Self-organization in social insects. *Trends in Ecology & Evolution*, 12(5):188–193, 1997.

[5] J. Carlson and R. R. Murphy. Reliability analysis of mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA'03*, volume 1, pages 274–281. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[6] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey. Sensor fault detection and identification in a mobile robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1383–1388. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[7] P. Goel, G. Dedeoglu, S. I. Roumeliotis, and G. S. Sukhatme. Fault detection and identification in a mobile robot using multiple model estimation and neural network. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA'00*, volume 3, pages 2302–2309. IEEE Computer Society Press, Los Alamitos, CA, 2000.

[8] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 531–537. AAAI Press/The MIT Press, Cambridge, MA, 2000.

[9] R. Dearden, F. Hutter, R. Simmons, S. Thrun, V. Verma, and T. Willeke. Real-time fault detection and situational awareness for rovers: Report on the Mars technology program task. In *Proceedings of IEEE Aerospace Conference*, volume 2, pages 826–840. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[10] V. Verma, G. Gordon, R. Simmons, and S. Thrun. Real-time fault diagnosis. *IEEE Robotics & Automation Magazine*, 11(2):56–66, 2004.

[11] P. Li and V. Kadirkamanathan. Particle filtering based likelihood ratio approach to fault diagnosis in nonlinear stochastic systems. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 31(3):337–343, 2001.

[12] V. Verma and R. Simmons. Scalable robot fault detection and identification. *Robotics and Autonomous Systems*, 54(2):184–191, 2006.

[13] R. Canham, A. H. Jackson, and A. Tyrrell. Robot error detection using an artificial immune system. In *Proceedings of NASA/DoD Conference on Evolvable Hardware, 2003*, pages 199–207. IEEE Computer Society, Washington, DC, 2003.

[14] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, 2008.

[15] L. E. Parker. ALLIANCE: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[16] B. P. Gerkey and M. J. Matarić. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

[17] B. P. Gerkey and M. J. Matarić. Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA'02*, pages 464 – 469. IEEE Press, Piscataway, NJ, 2002.

[18] M. B. Dias, M. B. Zinck, R. M. Zlot, and A. Stentz. Robust multirobot coordination in dynamic environments. In *Proceedings of IEEE Conference on Robotics and Automation, ICRA'04*, volume 4, pages 3435 – 3442. IEEE Press, Piscataway, NJ, 2004.

[19] M. A. Lewis and K. H. Tan. High precision formation control of mobile robots using virtual structures. *Autonomous Robots*, 4(4):387–403, 1997.

[20] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G. Theraula. *Self-Organization in Biological Systems*. Princeton University Press, NJ, 2001.

[21] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.

[22] J. Buck. Synchronous rhythmic flashing of fireflies. II. *Quarterly Review of Biology*, 63:265–289, 1988.

[23] H. M. Smith. Synchronous flashing of fireflies. *Science*, 82(2120):151–152, 1935.

[24] L. Glass. Synchronization and rhythmic processes in physiology. *Nature*, 410:277–284, 2001.

[25] W. A. Snedden, M. D. Greenfield, and Y. Jang. Mechanisms of selective attention in grasshopper choruses: who listens to whom? *Behavioral Ecology & Sociobiology*, 43(1):59–66, 1998.

[26] M. K. McClintock. Menstrual synchrony and suppression. *Nature*, 229(5282):244–245, 1971.

[27] Z. Néda, E. Ravasz, Y. Brechet, T. Vicsek, and A. L. Barabási. Self-organizing processes: The sound of many hands clapping. *Nature*, 403(6772):849–850, 2000.

[28] S. H. Strogatz. From Kuramoto to Crawford: exploring the onset of synchronization in populations of coupled oscillators. *Physica D*, 143(1-4):1–20, 2000.

[29] E. M. Izhikevich. Weakly pulse-coupled oscillators, FM interactions, synchronization, and oscillatory associative memory. *IEEE Transactions on Neural Networks*, 10(3):508–526, 1999.

[30] R. E. Mirollo and S. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[31] D. Lucarelli and I. J. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 62–68. ACM Press, New York, NY, 2004.

[32] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[33] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pages 1965–1970. IEEE Computer Society, Washington, DC, 2001.

[34] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor. Firefly-inspired heartbeat synchronization in overlay networks. In *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO'07*, pages 77–86. IEEE Computer Society Press, Los Alamitos, CA, 2007.

[35] A. Tyrrell and G. Auer. Imposing a reference timing onto firefly synchronization in wireless networks. In *Proceedings of the 65th IEEE Conference on Vehicular Technology, VTC2007*, pages 222–226. IEEE Computer Society Press, Los Alamitos, CA, 2007.

[36] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 142–153. ACM Press, New York, NY, 2005.

[37] X. Li and L. E. Parker. Sensor analysis for fault detection in tightly-coupled multi-robot team tasks. In *Proceedings of IEEE International Conference on Robotics and Automation, (ICRA'07)*, pages 3269–3276. IEEE Computer Society Press, Los Alamitos, CA, 2007.

[38] A. F. T. Winfield and J. Nembrini. Safety in numbers: fault-tolerance in robot swarms. *International Journal of Modelling, Identification and Control*, 1(1):30–37, 2006.

[39] A. F. T. Winfield, C. J. Harper, and J. Nembrini. Towards dependable swarms and a new discipline of swarm engineering. In *Swarm Robotics Workshop: State-of-the-art Survey*, pages 126–142. Springer Verlag, Berlin, Germany, 2005.

[40] F. Mondada, G. C. Pettinaro, A. Guignard, I. V. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo. SWARM-BOT: A new distributed robotic concept. *Autonomous Robots*, 17(2–3):193–221, 2004.

[41] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2–3):223–245, 2004.

[42] V. Trianni and M. Dorigo. Self-organisation and communication in groups of simulated and physical robots. *Biological Cybernetics*, 95:213–231, 2006.

[43] R Groß, M. Bonani, F. Mondada, and M. Dorigo. Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22(6):1115–1130, 2006.

[44] R. O'Grady, R. Groß, F. Mondada, M. Bonani, and M. Dorigo. Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. In *Advances in Artificial Life: 8th European Conference, ECAL 2005*, volume 3630, pages 272–281. Springer Verlag, Berlin, Germany, 2005.

[45] S. Nouyan, R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Group transport along a robot chain in a self-organised robot colony. In *Intelligent Autonomous Systems 9, IAS 9*, pages 433–442. IOS Press, Amsterdam, The Netherlands, 2006.

[46] A. L. Christensen. Efficient neuro-evolution of hole-avoidance and phototaxis for a swarm-bot. Technical Report TR/IRIDIA/2005-14, IRIDIA, Université Libre de Bruxelles, Belgium, October 2005. DEA Thesis.

[47] A. L. Christensen, R. O'Grady, and M. Dorigo. Photos and videos of firefly-inspired synchronization and fault tolerance experiments with robots, `http://iridia.ulb.ac.be/supp/IridiaSupp2008-003/`.

[48] R. Groß, E. Tuci, M. Dorigo, M. Bonani, and F. Mondada. Object transport by modular robots that self-assemble. In *Proceedings of the 2006 IEEE International Conferance on Robotics and Automation*, pages 2558–2564. IEEE Computer Society Press, Los Alamitos, CA, 2006.

View publication stats