From Fireflies to Fault-Tolerant Swarms of Robots

Anders Lyhne Christensen, Rehan O'Grady, and Marco Dorigo, *Fellow, IEEE*

***Abstract*—One of the essential benefits of swarm robotic 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 swarm robotic systems, few studies have been dedicated to the subject. In this paper, we take inspiration from the synchronized flashing behavior observed in some species of fireflies. We derive a completely decentralized algorithm to detect non-operational robots in a swarm robotic system. Each robot flashes by lighting up its on-board light-emitting diodes (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 swarm robotic 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.**

***Index Terms*—Fault detection, self-organization, swarm intelligence, swarm robotics, synchronization.**

I. Introduction

In this paper, we leverage some of the high-level principles behind synchronizing systems found in nature to obtain a robust, simple, decentralized approach to fault detection in groups or swarms of autonomous robots. By detecting the presence of 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.

Carlson *et al.* [1] tracked the reliability of 15 mobile robots from three different manufacturers over a period of 3 years and found the average mean time between failures to be 24 h. 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 multirobot systems.

Manuscript received January 15, 2008; revised June 17, 2008 and November 2, 2008; accepted February 4, 2009. First version published July 17, 2009; current version published August 14, 2009. 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).

A. L. Christensen is with the DCTI, Lisbon University Institute, Av. das Forças Armadas, 1649-026 Lisbon, Portugal. He is also with IRIDIA, CoDE, Université Libre de Bruxelles, 1050 Brussels, Belgium (e-mail: anders.christensen@iscte.pt)

R. O'Grady and Marco Dorigo are with the IRIDIA, CoDE, Université Libre de Bruxelles, 1050 Brussels, Belgium (e-mail: rogrady@ulb.ac.be, mdorigo@ulb.ac.be)

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2009.2017516

Many studies have been devoted to *endogenous fault detection*, that is, a robot detecting faults in itself, see for instance [2]–[10]. 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 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 render the robot unable to alert other robots or a human operator. The robustness of a multirobot system is therefore greater if the robots can detect the presence of faults in one another. We refer to this process as *exogenous fault detection*.

We are interested in exogenous fault detection in a particular type of multirobot systems, namely *swarm robotic systems*. A swarm robotic system is composed of numerous relatively simple robots. The philosophy behind swarm robotics is to rely on group-level self-organization through local interactions between the robots. The potential advantages of designs adhering to this philosophy include scalability, inherent parallelism, and robustness to individual failures [11]–[13].

In this paper, we present a simple and completely distributed approach that allows robots in a swarm robotic system to detect the presence of faults in one another: through local interactions, robots are able to synchronize and reach a state in which they flash their on-board light-emitting diode (LED) 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 Section II, we discuss previous studies related to fault detection and fault tolerance in multirobot systems. In Section III, we present the robotic hardware (the *swarm-bot* platform) used in this paper. In Section IV, we discuss different ways of detecting faults based on different flashing schemes, and we motivate our firefly-inspired approach. In Section V, we discuss synchronization in systems of pulse-coupled oscillators. In Section VI, 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 Section VII, we show how non-operational robots can be detected and present results of experiments with a system in which the robots have (simulated) self-repair capabilities. Concluding remarks and directions for future research are provided in Section VIII.

II. Related Work

For swarm robotic systems, little work on exogenous fault detection has been done. Previous studies have mostly been concerned with the analysis of a swarm's implicit tolerance to faults (as opposed to studying methods to explicitly detect and deal with faults). Lewis and Tan [14] have shown that their control algorithm for maintaining geometric formations exhibits correct behavior even if one of the robots fails. Winfield and Nembrini [15] analyzed potential faults and their effect in 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 conclusions, they envisage a new behavior in which the swarm can identify failed members and isolate them from the rest of the swarm. In both [14] and [15] 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 [16]), it is not known whether such designs are feasible (or even theoretically possible) for all systems and all tasks.

Fault detection and fault tolerance in systems composed of a limited number of relatively complex robots have, however, been studied in the past. Parker [17] has, for instance, demonstrated that cooperating teams of robots based on the ALLIANCE software architecture can achieve a high degree of fault tolerance. ALLIANCE was developed for heterogeneous robots performing missions composed of loosely coupled tasks with possible ordering dependencies. Fault tolerance is obtained by modeling "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 to perform 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 [18], [19] and TraderBots [20], 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.

The objective of approaches such as ALLIANCE, MURDOCH, and TraderBots is to enable a team of robots to coordinate their efforts effectively—even when faults are present. These approaches require that the robots are tightly coupled. For multirobot systems consisting of a limited number of relatively sophisticated robots, these approaches have proved to be practical. We are, however, interested in multirobot systems of the swarm robotic variety. For swarm robotic systems, tightly coupled and/or centralized approaches are less easily applicable. Robots in such systems tend to lack the means to facilitate centralized swarm-level coordination. Furthermore, both tightly coupled and centralized approaches often have serious scalability issues.

In this paper, we are interested in a fault detection approach that scales up to hundreds of relatively simple robots. Therefore, we propose an approach that is completely decentralized and in which each robot uses only local information.

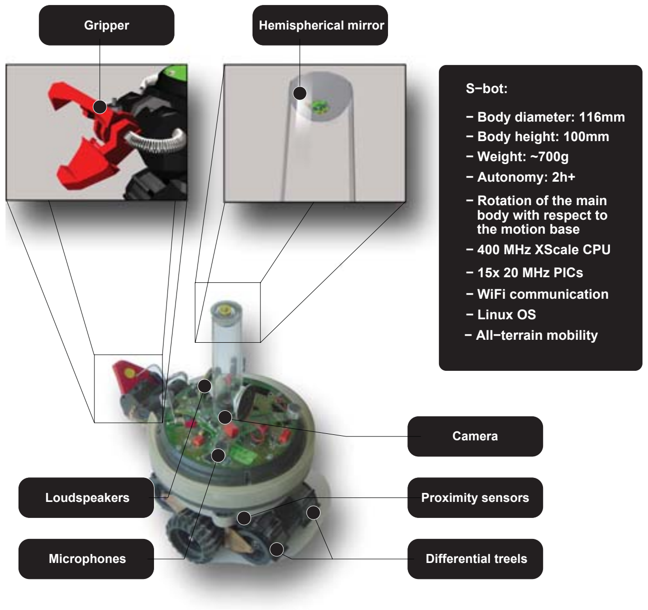

Fig. 1. *s-bot*: an autonomous, mobile robot capable of forming physical connections with other *s-bots*.

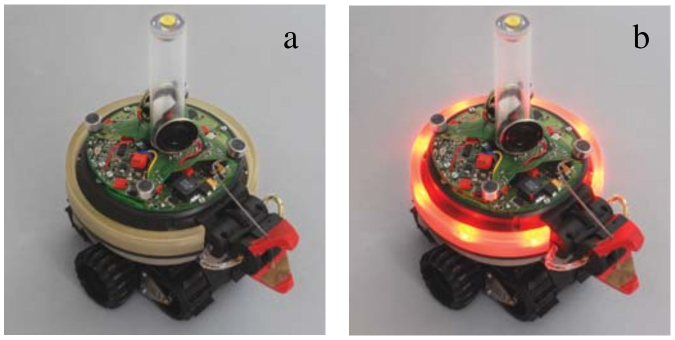

Fig. 2. a) *s-bot* with all its LEDs off. b) *s-bot* with its red LEDs illuminated.

III. Hardware Platform

For our experiments, we use the *swarm-bot* robotic platform [21]. This innovative platform was designed and built by Francesco Mondada's group at the Laboratoire de Système Robotiques (LSRO) of the Ecole Polytechnique Fédérale de Lausanne (EPFL). The system consists of a number of mobile autonomous robots called *s-bots* (see Fig. 1) 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 [22], [23]). 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 [24]–[27].

Each *s-bot* is equipped with an XScale CPU running at 400-MHz, a number of sensors including an omni-directional camera, light sensors, 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. 2).

Each *s-bot* control program operates 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 paper, we use a control cycle period of 0.15 s.

A. S-bot Camera and Image Processing

The *s-bots* are equipped with an omnidirectional camera mounted in the turret. The camera points upward at a hemispherical mirror mounted in a transparent perspex tube above the turret. The *s-bot* camera captures images of the robot's surroundings reflected in the hemispherical mirror. The camera sensor records 640×480 pixel 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 multipixel 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 our experiments, robots always illuminate their green LEDs, except when they are flashing, to ensure that nearby robots are always able to see one another.

B. Simulation Environment

We use a custom software simulator [28]. All of the sensors and actuators shown in Figs. 1 and 2 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 are available 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.

IV. Motivation and Method

In this section, we discuss the motivation for a protocol in which *s-bots* periodically and in unison illuminate their on-board LEDs in a certain color. We first explain why color-based communication is a sensible communication modality to use for exogenous fault detection. We then go on to discuss three different protocols for exogenous fault detection based on visual color-based communication.

Robots can detect the presence of faults in one another when they periodically send out an "I am operational" signal. By detecting the absence of this signal over a certain time period, team members can detect non-operational robots. On the *s-bots*, we have a number of options concerning the medium for the signal: radio communication, sound communication, and visual color-based communication. Over a radio network, robots could periodically send out broadcast messages to indicate that they are still operational. If sound communication is used, the robots could periodically emit a tone with a certain frequency. If visual communication is used, *s-bots* could periodically illuminate their on-board LEDs in a certain color.

When robots detect the presence of an exogenous fault, it is important that they at the same time can detect the location of the failed robot. To this end, the communication between robots should be *situated* rather than *abstract* [29]. Both radio and sound often provide only abstract communication: when one robot sends out a message, other robots may receive the message but they are unable to determine the location of the sender relative to their own frame of reference. Thus, the messages are separated from environment localization information. In some cases, radio and sound can provide some level of situated communication: certain radio-based technologies allow robots to confine communication to short distances. Furthermore, it may be possible to obtain a crude approximation of the direction and distance to the emitting source if directional antennas are used. Such hardware is, however, rarely available on current robotic platforms. Alternatively, using sound-based communication in a system where each robot is equipped with multiple microphones, a robot can with reasonable accuracy detect the direction of a sound emitter using time delay estimates. However, estimating the distance with a sufficiently high accuracy can be very difficult.

Visual communication is, on the other hand, implicitly situated: robots detect changes in each other's LED colors by analyzing camera images. As a direct result of this analysis, the perceiving robot knows the location of any LEDs it detects in its own frame of reference. Some other multirobot systems have different situated communication modalities. On iRobot's Swarm platform, for instance, each robot can estimate the range and bearing of neighboring robots with a high level of accuracy using a communication system based on multiple infrared transceivers [30]. Such hardware is, however, not available on our platform. We therefore chose to use visual communication.

In the following sections, we justify our so called *synchronized flashing protocol* by comparing it with two other possible visual communication-based protocols, namely the *unsynchronized flashing protocol* and the *ping-pong protocol*.

A. Unsynchronized Flashing Protocol

In this protocol, either the red LEDs or the green LEDs of all the *s-bots* are constantly illuminated. Each *s-bot* periodically switches the LED color that it is illuminating. The *s-bots* share a common system-wide illumination period, but make no effort to synchronize their switching. Using a common illumination period of 2 s, for example, each *s-bot* would illuminate its green LEDs for 2 s, then its red LEDs for 2 s, then its green LEDs for 2 s and so on. A robot that does not change the color of its LEDs may have a fault.

The unsynchronized flashing protocol requires that every robot continuously monitors all its neighboring robots. Keeping track of individual robots is hard. The image processing capabilities of the *s-bots* are limited: the *s-bots* can see only illuminated LEDs. Features such as the gripper, the treels, and chassis cannot be identified by the image processor. Each LED is detected as one or more distinct objects. One robot can partly or completely occlude another robot. When a robot moves around, the perspex tube holding the camera and the hemispherical mirror (see Fig. 1) shakes and as a result even static objects "jump around" from frame to frame. Hence, it can be difficult for a robot to determine which LEDs belong to which robot, especially when two or more neighboring robots are close to one another. Furthermore, due to the robots' limited visual range, they enter and exit each other's view repeatedly. The difficulty of identifying and keeping track of moving robots makes the *unsynchronized flashing protocol* hard to implement in practice.

B. Ping-Pong Protocol

Assume that each robot with a small probability sends out a message (a *ping*) to which neighboring robots should respond (with a *pong*). If a robot does not reply, it may have a fault. In the following discussion, we assume that a robot sends out a ping by illuminating its red LEDs, while other robots respond with a pong by illuminating their green LEDs.

In the ping-pong protocol, the pinging robot (a robot that illuminates its red LEDs for a brief period) has to ensure that all of its neighboring robots correctly respond with a pong (by illuminating their green LEDs for a brief period). Therefore, as with the unsynchronized flashing protocol, a robot using the ping-pong protocol needs to keep track of neighboring robots. However, the ping-pong protocol has the advantage that a robot only needs to keep track of its neighbors right after it has sent out a ping. When a robot sends out a ping, it can stop its motion so that the camera and perspex tube do not move. In this way, the *s-bot* can see more accurately whether its neighboring robots respond correctly. The issue of keeping track of neighboring robots is therefore not as pronounced in the ping-pong protocol as it is in the unsynchronized flashing protocol. However, the following two problematic situations can arise: A robot (N) does not see a ping from another robot (P), due, for instance, to occlusion or to the distance between the two robots. P, however, is able to see some of Ns LEDs while waiting for its neighbors to respond with a pong: P would wrongly assume that N has a fault giving that N does not reply to the ping that it did not see. Another situation can arise in which a robot receives ping messages from different robots in succession and therefore illuminates its green LEDs for an extended period. A robot that is merely responding to a series of pings is hard to distinguish from a robot that has become non-operational while it was replying.

C. Synchronized Flashing Protocol

The problems associated with the two protocols above disappear if we assume that the robots change color in a synchronized fashion. In the synchronized flashing protocol, any robot that is not flashing when the rest of the swarm flashes may have a fault. In both the unsynchronized flashing protocol and the ping-pong protocol, robots need to correlate visual information distributed in time in order to determine whether a robot is faulty. In contrast, when robots flash in unison, there is no need for a single robot to identify and track all of its neighbors. It is sufficient for a robot to detect LEDs with the wrong color (which indicates that one of its neighbors is not flashing) while the robot itself is flashing. Hence, the presence of a potentially failed robot can be detected on the basis of a single image captured by the camera. More complex reasoning required to keep track of all of the neighboring robots over time is not necessary. Furthermore, the two issues listed above for the ping-pong protocol (unseen pings and constantly replying to pings) do not arise in the synchronized flashing protocol: since the robots flash in synchrony, every robot is in effect "pinging" and "ponging" at the same time.

The challenge in implementing a synchronized flashing protocol is to make all the robots change color periodically at the same time. However, nature has found an elegant solution to this problem.

V. Synchronization of Pulse-Coupled Oscillators

In nature, we find many examples of coupled oscillating systems with 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 [31], [32]. Other examples include cardiac cells [33], choruses of grasshoppers [34], female menstrual cycles [35], and clapping in theaters [36].

Systems of coupled oscillators can be divided into two classes: oscillators that continuously influence one another (see for instance [37]) and so called *integrate-and-fire* or *pulse-coupled* oscillators, where one oscillator only influences other oscillators during short, periodic pulses. In this paper, 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 its 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 [38]:

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

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 distribution function $\delta(t)$ satisfies $\delta = 0$ 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. 3.

The self-synchronization of pulse-coupled oscillators was first described by Peskin [39], who observed the phenomenon in cardiac pacemaker cells. Mirollo and Strogatz later showed that a population of fully connected pulse-coupled oscillators almost always evolves to a state in which all oscillators are

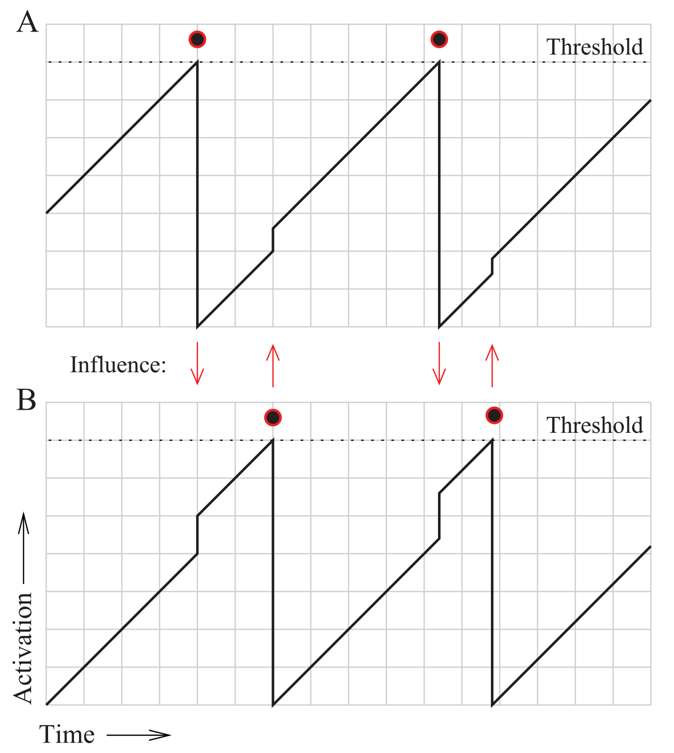

Fig. 3. Example of two pulse-coupled oscillators. Both oscillators increase at a constant rate until the threshold is reached—at which point the oscillator fires and jumps back to 0—or until the firing of the other oscillator is observed. When one oscillator observes the other's firing, it increases 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.

firing synchronously [40]. Recently, Lucarelli and Wang [41] 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 [42], [43]). The behavior of fireflies has inspired algorithms for heartbeat synchronization in overlay networks [44], imposing reference timing in wireless networks [45], and in sensor networks for coordinating sensing and communication [46].

In this paper, 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.

VI. Synchronization in Robots

We let each robot act as an integrate-and-fire oscillator. When the activation of the oscillator reaches a certain threshold, the robot illuminates its red LEDs as in the example shown in Fig. 2 and resets its activation. When neighboring robots (within 50 cm) detect the flash, they increment their own activation.

[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.

A. Discrete Oscillators

Due to the inherent discreteness of the sense-think-act control paradigm, we transform the continuous model in (1) into a discrete model with piece-wise linear dynamics

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

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 paper, 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 Section VI-B. $\alpha_i(n)$ is the number of flashing robots seen by i in control cycle n. We use the linear pulse-coupling function

$$h(x) = x. \qquad (3)$$

When $x_i(n)$ exceeds 1, robot i flashes and its activation is reset to 0. There is a (small) latency from the moment when the control program sends a signal to the flash LEDs 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 five 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 five 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 $\alpha_i(n)$ in the following way. We dissect a robot's omni-directional field of view into 16 equally sized slices and count only flashes from slices from which no flashes were perceived in the previous control cycle. In this way, we obtain a reasonable estimate of the number of flashing robots without the need to identify and track individual robots (as discussed in Section IV, it is difficult for one robot to keep track of all of its neighboring robots).

B. Synchronization Experiments in Simulation

We are interested in the time it takes for our swarm robotic system to synchronize. We define the system to be synchronized when it is in a state where the value of every activation $x_i(n)$ is no further than $1/T$ from all other activations. An example of the development of activation values sampled every Tth control cycle during a run with 25 robots in simulation is shown in Fig. 4. The activation for each robot at every Tth

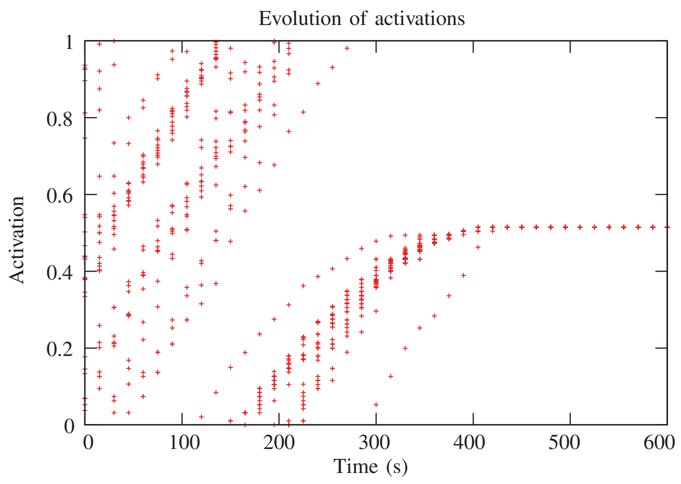

Fig. 4. Example of the evolution of activations sampled every Tth control cycle in 25 mobile robots over the course of 10 min. One cross represents the activation for a single robot at the corresponding time.

control cycle 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 defines 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 time it takes for swarms of 10, 25, 50, and 100 robots to synchronize (see Fig. 5), at densities of 2, 4, 6, 8, and 10 robots/m^2 in square-shaped arenas with 50 simulated robots (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 time 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 are shown in Fig. 8. When the flash wave starts, all the robots have activations close to the firing threshold 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

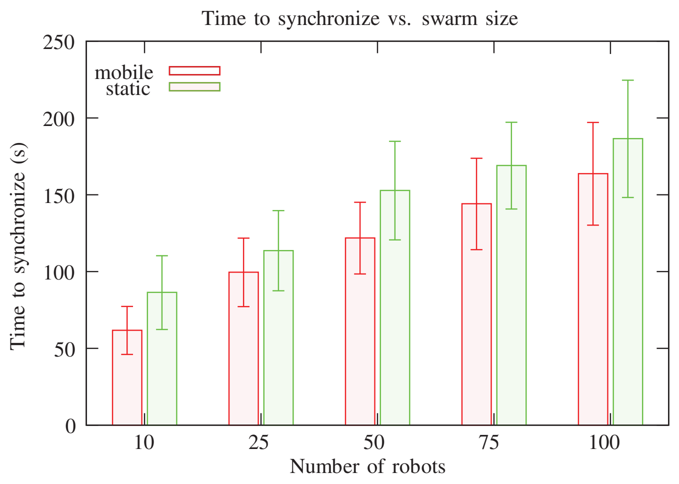

Fig. 5. Synchronization time for groups of 10 to 100 simulated robots. Each bar summarizes 100 runs and error bars denote the standard deviation. The density was 8 robots/m^2 and a coupling constant of $\epsilon = 0.1$ was used in all runs.

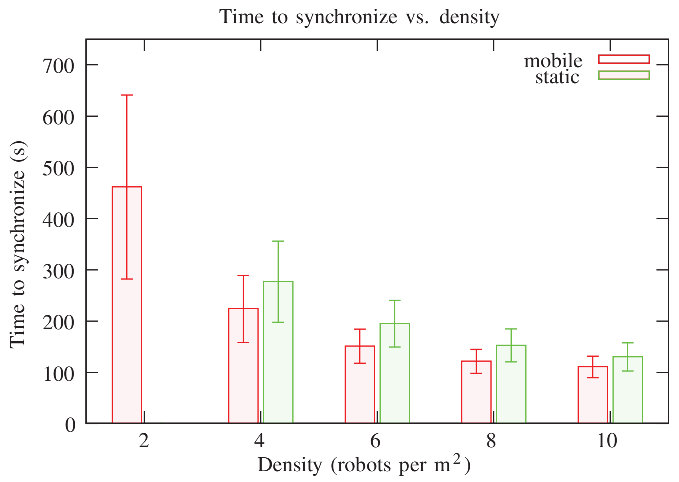

Fig. 6. Synchronization time for a group of 50 simulated robots at different densities. Each bar summarizes 100 runs and error bars denote the standard deviation. A coupling constant of $\epsilon = 0.1$ was used in all runs. Due to the limited sensory range of the *s-bots* (up to 50 cm), experiments 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.

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.

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.

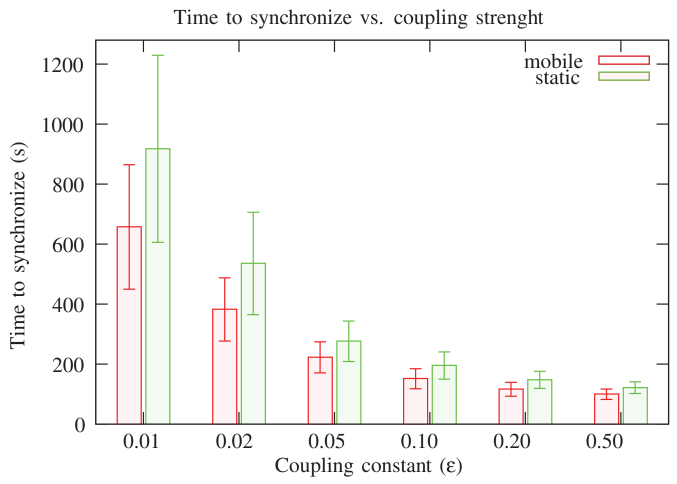

Fig. 7. Synchronization time for group of 50 simulated robots for coupling constants 0.01, 0.02, 0.05, 0.10, 0.20, 0.50. Each bar summarizes 100 runs and error bars denote the standard deviation. The density was 6 robots/m^2.

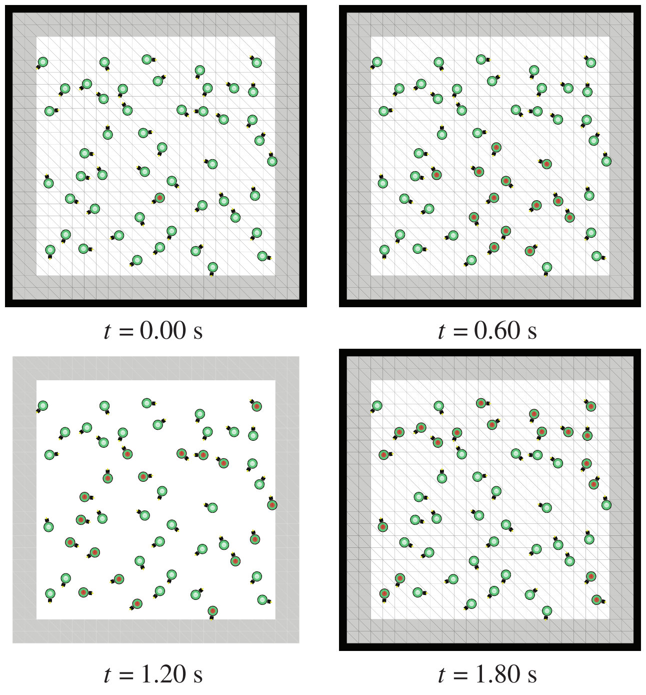

Fig. 8. Example of a flash wave in a group of static robots.

The synchronization time is linearly proportional (with a gentle slope) with swarm sizes up to 100 *s-bots* (see Fig. 5). The mean synchronization time for a group of 10 *s-bots* is 62 s, while for 100 *s-bots* the mean synchronization time is 164 s—less than three times slower.

The synchronization time at different robot 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 fault, such as a control program crash, that may cause it to become non-operational with its flash LEDs 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.

TABLE I

Synchronization Time for 10 Real Robots. Corresponding Results for Simulated Robots Are Shown in Brackets

	Mean (s)	St.dev. (s)	Shortest (s)	Longest (s)
Static	94	72	55	174
	(91)	(30)	(39)	(224)
Moving	77	28	30	118
	(71)	(28)	(27)	(158)

An interesting question is how the presence of obstacles affects the synchronization time for a swarm of robots. In an additional set of experiments, we evaluated the synchronization time for a swarm of moving robots in an environment containing obstacles and a low density of robots (2 robots/m^2). We used a 5 m × 5 m arena. The arena was divided into two rooms connected by a narrow corridor (50 cm wide), see Fig. 9. Both rooms contained two rectangular obstacles measuring 100 cm × 50 cm each. We conducted 100 trials with 50 simulated robots and with $\epsilon = 0.1$. In every trial, all the robots synchronized. The average synchronization time was 779 s (st. dev.: 343 s). The synchronization time for 50 simulated robots in an arena of the same dimension with no obstacles was 462 s (st. dev.: 180 s, see Fig. 6). A swarm of robots is thus able to synchronize even when obstacles are present although the synchronization process takes longer than if no obstacles are present.

C. 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. The experiments were performed in a walled arena with dimensions 1.6 m × 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 Section VI-B). Based on video recordings, the synchronization time 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. The experimental setups with static robots and with moving robots, respectively, were replicated 10 times with different initial conditions. Table I reports the results obtained in the real robot experiments, as well as the results obtained in simulation (same experimental conditions, 100 replications). Videos of the experiments can be downloaded from [47].

On real robots, we observe the same trend as in simulation: moving robots tend to synchronize faster than static robots. The mean synchronization time of static robots was 94 s, while

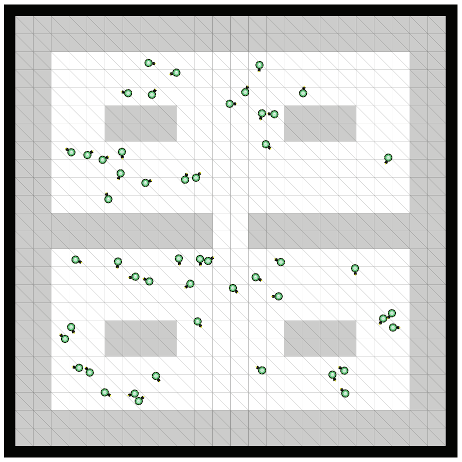

Fig. 9. Arena containing obstacles.

the mean synchronization time 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 sensor and actuator systems.

VII. 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 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. Firstly, it cannot be assumed that the robots are always synchronized. Secondly, the sensory range of the robots is limited.

A. Detecting Faults in Nonsynchronized 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

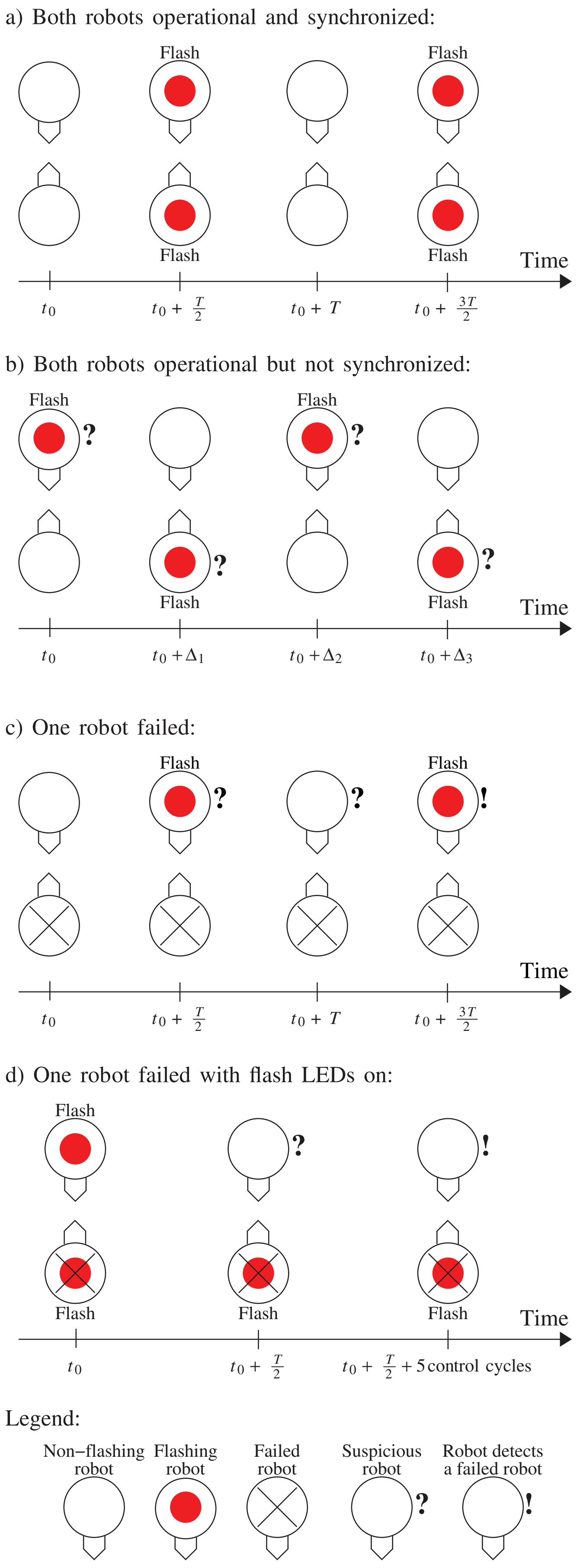

Fig. 10. Four possible scenarios. See text.

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. Under these circumstances, R2's second flash will often provoke R1 to flash. R2 can, in fact, calculate the sufficient conditions under which its 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

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

That is, if R2 flashes twice while R1 does not flash at all (including not being provoked to flash by R2's second flash) and if (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.

B. Limited Sensory Range

Since the robots are mobile and 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 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]

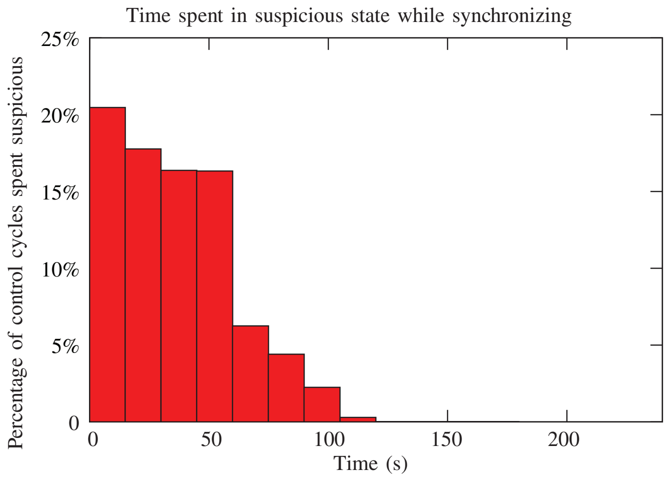

Fig. 11. Average percentage of the control cycles spent in the suspicious state over intervals of 15 s during a run with 50 simulated robots in a 2.5 m × 2.5 m arena. The robots were not initially synchronized.

C. 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 their task and wait while they determine whether the candidate robot is non-operational or whether 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 spent being suspicious in the beginning of an example run.

The time initially spent by the robots 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 tradeoff between the length of the warm-up period and the latency of fault detection since faults cannot be detected during the warm-up period.

D. Implementation

An overview of the control and fault detection logic executed every control cycle is shown in Algorithm 1. The activation x is incremented by the sum of the constant

[2]For simplicity we assume that the presence of a fault causes a robot to stop its movement. If this were not the case, then a suspicious robot 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.

Algorithm 1: ControlCycle()

```
ReadSensors();
if HasFlashed($candidate$) then
    $candidate = none$;
end
$x \leftarrow x + \frac{1}{T} + \epsilon\alpha h(x)$;
$\Delta \leftarrow \Delta + \epsilon\alpha h(x)$;
if $x > 1$ then
    FlashAndCheckForFailedRobots();
    $x \leftarrow 0$;
end
if $x$ has passed 0.5 then
    CheckForFailedRobotsWithFlashOn();
end
if $candidate \neq none$ then
    StopMoving();
else
    RandomWalkAndAvoidObstacles();
end
```

Algorithm 2: FlashAndCheckForFailedRobots()

```
TurnOnFlashLeds(5 cycles);
if $candidate = none$ then
    ...In case two or more neighboring robots are not flashing,
    ...the position of the closer one is returned:
    $candidate$ = CheckForNonFlashingRobots();
    $\Delta = 0$;
else
    if $\epsilon h(1 - \Delta) \geq \Delta$ then
        $failedrobot \leftarrow candidate$;
        ...A fault has been detected. Take
        ...actions to deal with the fault.
    else
        ...Wait until next flash before concluding
        ...whether the $candidate$ robot has failed or not.
    end
end
```

increase $1/T$ and the product of the coupling strength ϵ, the number of flashes seen α, and the pulse-coupling function $h(x)$. When x exceeds 1, the robot flashes and checks for non-flashing robots, while if x has just passed 0.5, a check is made to determine whether any neighboring robot 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 Algorithm 2 and Algorithm 3, respectively.

The current implementation of the *s-bot* vision software only allows the *s-bots* to see objects that display illuminated LEDs. In our experiments, the *s-bots* therefore always illuminate their on-board LEDs in some color—in red to indicate that they are flashing and in green in order to be visible to other robots while they are not flashing. Some faults, such as a dead battery, would cause an *s-bot* to go dark, that is, since the robot would no longer display any LEDs it would not be visible to the other *s-bots*. If we assume that operational robots have their LEDs illuminated in some color and if dark *s-bots* could be detected in some way (e.g., via proximity sensors), faults causing an *s-bot* to turn dark could be easily detected: the absence of any illuminated LEDs would immediately indicate that the dark *s-bot* had become non-operational. In this paper,

Algorithm 3: CheckForFailedRobotsWithFlashOn()

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

however, we have not implemented any logic to detect dark *s-bots*.

E. 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 toward a scenario in which robots can repair one another—either directly or by physically connecting and transporting failed robots[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 synchronization experiments for mobile robots described in Section VI-C (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 fault in one of the robots. The fault caused the robot to stop its movement and prevented it from changing the color of its LEDs. 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 Table II.

The presence of 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 attempted to grasp the failed robot twice, unsuccessfully on both occasions. Eventually another operational robot detected the fault and connected to the failed robot.

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, another 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.

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

TABLE II
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

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

During the experiment, one of the robots experienced a real hardware I/O fault similar to the real fault described above in Section VII-E. 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 righted a robot after it had toppled over due to a collision with two other robots. A video of the experiment can be found on [47].

These results suggest that our approach is robust even when several robots are experiencing faults at the same time. Furthermore, the results indicate that a swarm of robots can survive a relatively high rate of failure if the robots are able to repair one another.

VIII. CONCLUSION AND FUTURE WORK

In this paper, 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 synchronization time 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 actively 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.

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 time it takes for a swarm to synchronize depends on its size. 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 therefore 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 Section VI-B).

A potential direction for future research is to implement and evaluate 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 intelligence approach.

ACKNOWLEDGMENT

This paper would not have been possible without the innovative robotic hardware developed by F. Mondada's group at the Laboratoire de Système Robotiques of EPFL. 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. M. Dorigo acknowledges support from the F.R.S.-FNRS, of which he is a Research Director.

REFERENCES

[1] J. Carlson, R. R. Murphy, and A. Nelson, "Follow-up analysis of mobile robot failures," in *Proc. 2004 IEEE Int. Conf. Robotics Automat.*, Los Alamitos, CA: IEEE Computer Society Press, pp. 4987–4994.

[2] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey, "Sensor fault detection and identification in a mobile robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, vol. 3. Los Alamitos, CA: IEEE Computer Society Press, 1998, pp. 1383–1388.

[3] 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 *Proc. IEEE Int. Conf. Robotics Automat. '00*, vol. 3. Los Alamitos, CA: IEEE Computer Society Press, pp. 2302–2309.

[4] U. Lerner, R. Parr, D. Koller, and G. Biswas, "Bayesian fault detection and diagnosis in dynamic systems," in *Proc. 7th Nat. Conf. Artificial Intell.*, Cambridge, MA: AAAI Press/The MIT Press, 2000, pp. 531–537.

[5] 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 *Proc. IEEE Aerospace Conf.*, vol. 2. Los Alamitos, CA: IEEE Computer Society Press, 2004, pp. 826–840.

[6] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis," *IEEE Robot. Automat. Mag.*, vol. 11, no. 2, pp. 56–66, Jun. 2004.

[7] P. Li and V. Kadirkamanathan, "Particle filtering based like lihood ratio approach to fault diagnosis in nonlinear stochastic systems," *IEEE Trans. Syst., Man Cybern., Part C*, vol. 31, no. 3, pp. 337–343, Mar. 2001.

[8] V. Verma and R. Simmons, "Scalable robot fault detection and identification," *Robotics Autonomous Syst.*, vol. 54, no. 2, pp. 184–191, Feb. 2006.

[9] R. Canham, A. H. Jackson, and A. Tyrrell, "Robot error detection using an artificial immune system," in *Proc. NASA/DoD Conf. Evolvable Hardware, 2003*, Washington, D.C. IEEE Computer Society, pp. 199–207.

[10] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, no. 1, pp. 49–67, Jan. 2008.

[11] Y. U. Cao, A. S. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, Mar. 1997.

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intell.: From Natural to Artificial Syst.*. New York: Oxford Univ. Press, 1999.

[13] W.-M. Shen, P. Will, A. Galstyan, and C. M. Chuong, "Hormone-inspired self-organization and distributed control of robotic swarms," *Autonomous Robots*, vol. 17, no. 1, pp. 93–105, Jul. 2004.

[14] M. A. Lewis and K. H. Tan. "High precision formation control of mobile robots using virtual structures," *Autonomous Robots*, vol. 4, no. 4, pp. 387–403, Oct. 1997.

[15] A. F. T. Winfeld and J. Nembrini, "Safety in numbers: Fault-tolerance in robot swarms," *Int. J. Model., Identificatin Control*, vol. 1, no. 1, pp. 30–37, Jan. 2006.

[16] A. F. T. Winfeld, C. J. Harper, and J. Nembrini, "Toward dependable swarms and a new discipline of swarm engineering," in *Swarm Robotics Workshop: State-of-the-art Survey*, Berlin, Germany: Springer Verlag, 2005, pp. 126–142.

[17] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Trans. Robotics Automat.*, vol. 14, no. 2, pp. 220–240, Apr. 1998.

[18] B. P. Gerkey and M. J. Matarić, "Sold!: Auction methods for multirobot coordination," *IEEE Trans. Robotics Automat.*, vol. 18, no. 5, pp. 758–768, Oct. 2002.

[19] B. P. Gerkey and M. J. Mataric, "Pusher-watcher: An approach to fault-tolerant tightly coupled robot coordination," in *Proc. IEEE Int. Conf. Robotics Automat. '02*. Piscataway, NJ: IEEE Press, pp. 464–469.

[20] M. B. Dias, M. B. Zinck, R. M. Zlot, and A. Stentz, "Robust multirobot coordination in dynamic environments," in *Proc. IEEE Int. Conf. Robotics Automat. '04*, vol. 4. Piscataway, NJ: IEEE Press, pp. 3435–3442.

[21] F. Mondada, G. C. Pettinaro, A. Guignard, I. V. Kwee, D. Floreano, J.-L. Deneubourg, S. Nol, L. M. Gambardella, and M. Dorigo, "SWARM-BOT: A new distributed robotic concept," *Autonomous Robots*, vol. 17, no. 2–3, pp. 193–221, Sep. 2004.

[22] M. Dorigo, V. Trianni, E. Şahin, R. Gross, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella, "Evolving self-organizing behaviors for a swarm-bot," *Autonomous Robots*, vol. 17, no. 2–3 pp. 223–245, 2004.

[23] V. Trianni and M. Dorigo, "Self-organization and communication in groups of simulated and physical robots," *Biol. Cybern.*, vol. 95, no. 3, pp. 213–231, Aug. 2006.

[24] R. Gross, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous self-assembly in swarm-bots," *IEEE Trans. Robotics*, vol. 22, no. 6, pp. 1115–1130, Dec. 2006.

[25] 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 *Proc. Adv. Artificial Life: 8th Eur. Conf. ECAL 2005*, vol. 3630. Berlin, Germany: Springer Verlag, pp. 272–281.

[26] S. Nouyan, A. Campo, and M. Dorigo, "Path formation in a robot swarm: Self-organized strategies to find your way home," *Swarm Intell.*, vol. 2, no. 1, pp. 1–23, Mar. 2008.

[27] S. Nouyan, R. Gross, M. Bonani, F. Mondada, and M. Dorigo, "Teamwork in self-organized robot colonies," *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, Aug. 2009.

[28] A. L. Christensen, "Efficient neuro-evolution of hole-avoidance and phototaxis for a swarm-bot," Memoire de DEA, Université Libre de Bruxelles, Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2005-14, 2005.

[29] K. Støy, "Using situated communication in distributed autonomous mobile robots," in *Proc. 7th Scandinavian Conf. Artificial Intell.*, Amsterdam, Netherlands: IOS Press, 2001, pp. 44–52.

[30] J. McLurkin and J. Smith, "Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots," in *Proc. 7th Int. Symp. Distributed Autonomous Robotic Syst.*, Berlin, Germany: Springer Verlag, 2004, pp. 399–408.

[31] J. Buck, "Synchronous rhythmic flashing of fireflies II," *Quart. Rev. Biol.*, vol. 63, no. 3, pp. 265–289, Sep. 1988.

[32] H. M. Smith, "Synchronous flashing of fireflies," *Science*, vol. 82, no. 2120, pp. 151–152, Aug. 1935.

[33] L. Glass, "Synchronization and rhythmic processes in physiology," *Nature*, vol. 410, no. 6825, pp. 277–284, Mar. 2001.

[34] W. A. Snedden, M. D. Greenfield, and Y. Jang, "Mechanisms of selective attention in grasshopper choruses: Who listens to whom?" *Behavioral Ecology and Sociobiol.*, vol. 43, no. 1, pp. 59–66, Jun. 1998.

[35] M. K. McClintock, "Menstrual synchrony and suppression," *Nature*, vol. 229, no. 5282, pp. 244–245, Jan. 1971.

[36] Z. Néda, E. Ravasz, Y. Brechet, T. Vicsek, and A. L. Barabasi, "Self-organizing processes: The sound of many hands clapping," *Nature*, vol. 403, no. 6772, pp. 849–850, Feb. 2000.

[37] S. H. Strogatz, "From Kuramoto to Crawford: Exploring the onset of synchronization in populations of coupled oscillators, *Physica D*, vol. 143, no. 1–4, pp. 1–20, Sep. 2000.

[38] E. M. Izhikevich, "Weakly pulse-coupled oscillators, FM interactions, synchronization, and oscillatory associative memory," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 508–526, May 1999.

[39] C. S. Peskin, *Mathematical Aspects Heart Physiology*. New York: Courant Institute of Mathemaical Sciences, New York Univ., 1975.

[40] R. E. Mirollo and S. H. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM J. Appl. Math.*, vol. 50, no. 6, pp. 1645–1662, Dec. 1990.

[41] D. Lucarelli and I. J. Wang, "Decentralized synchronization protocols with nearest neighbor communication," in *Proc. 2nd Int. Conf. Embedded Networked Sensor Syst.*, New York: ACM, 2004, pp. 62–68.

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

[43] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proc. 15th Int. Parallel Distributed Process. Symp.*, Washington, D.C.: IEEE Computer Society, 2001, pp. 1965–1970.

[44] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor, "Firefly- inspired heartbeat synchronization in overlay networks," in *Proc. 1st Int. Conf. Self-Adaptive Self-Organizing Syst.'07*, Los Alamitos, CA: IEEE Computer Society Press, pp. 77–86.

[45] A. Tyrrell and G. Auer, "Imposing a reference timing onto firefly synchronization in wireless networks," in *Proc. 65th IEEE Conf. Vehicular Technology VTC 2007*," Los Alamitos, CA: IEEE Computer Society Press, pp. 222–226.

[46] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proc. 3rd Int. Conf. Embedded Networked Sensor Systems*, New York: ACM, 2005, pp. 142–153.

[47] A. L. Christensen, R. O'Grady, and M. Dorigo. (2009, Jun. 25). *Photos and videos of firefly-inspired synchronization and fault tolerance experiments with robots* [Online]. Available: 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 *Proc. 2006 IEEE Int. Conf. Robotics Automat.*, Los Alamtos, CA: IEEE Computer Society Press, pp. 2558–2564.

Anders Lyhne Christensen obtained the Master's degree in computer science and bioinformatics from Aalborg University, Denmark, in 2002. Later that year, he received a Marie Curie Fellowship hosted by the Dependable Systems Group at the Universidade de Coimbra and Critical Software, Portugal. He completed the Diplôme d'Études Approfondies (Master's equivalent) at the Université Libre de Bruxelles, Brussels, Belgium, in 2005. In 2008, he obtained the Ph.D. degree from IRIDIA-CoDE, Université Libre de Bruxelles, Belgium.

He is currently an Assistant Professor at DCTI, Lisbon University Institute, Portugal. He has spent several years in the private sector and on software projects ranging from 3-D acoustics to high-performance computing. He has published work in bioinformatics, high-performance computing, and autonomous robotics. His current research interests are in dependable swarm robotics and autonomous self-assembly.

Rehan O'Grady graduated in 1999 from Edinburgh University with First Class Honours in mathematics and computer science. He received the Diplôme d'´Etudes Approfondies (Master's equivalent) from the Université Libre de Bruxelles, Brussels, Belgium, in 2005.

He is currently a researcher at the IRIDIA-CoDE laboratory of Université Libre de Bruxelles, Brussels, Belgium. He is a recipient of Kevin Clark memorial prize, awarded each year to the top mathematics and computer science graduate at Edinburgh University. He was in the software industry for several years. During his time at Micromuse Plc., he developed a system for monitoring usage outages in internet network services, which was subsequently patented. His current research interests are in swarm robotics and self-assembling robotic systems.

Marco Dorigo (S'92–M'93–SM'96–F'06) received the Laurea (Master of Technology) degree in industrial technologies engineering in 1986 and the doctoral degree in information systems and electronic engineering in 1992 from Politecnico di Milano, Milan, Italy, and the title of Agrégé de l'Enseignement Supérieur, from the Université Libre de Bruxelles, Belgium, in 1995.

From 1992 to 1993, he was a Research Fellow at the International Computer Science Institute of Berkeley. In 1993, he was a NATO-CNR Fellow, and from 1994 to 1996, a Marie Curie Fellow. Since 1996, he has been a tenured researcher of the FNRS, the Belgian National Funds for Scientific Research, and a Research Director of IRIDIA-CoDE, the artificial intelligence laboratory of Université Libre de Bruxelles, Brussels, Belgium. He is the inventor of the ant colony optimization meta-heuristic. His current research interests include swarm intelligence, swarm robotics, and metaheuristics for discrete optimization.

Dr. Dorigo was awarded the Italian Prize for Artificial Intelligence in 1996, the Marie Curie Excellence Award in 2003, the Dr. A. De Leeuw-Damry-Bourlart award in applied sciences in 2005, and the Cajastur International Prize for Soft Computing in 2007. He is the Editor-in-Chief of *Swarm Intelligence* and an Associate Editor or Member of the Editorial Board for many journals in computational intelligence and adaptive systems. He is a fellow of the ECCAI.