Bioinspiration &
Biomimetics

PAPER • **OPEN ACCESS**

A swarm design paradigm unifying swarm behaviors using minimalistic communication

To cite this article: Joshua Cherian Varughese *et al* 2020 *Bioinspir. Biomim.* **15** 036005

Bioinspiration & Biomimetics

OPEN ACCESS

RECEIVED
22 October 2019

REVISED
24 December 2019

ACCEPTED FOR PUBLICATION
22 January 2020

PUBLISHED
5 March 2020

PAPER

A swarm design paradigm unifying swarm behaviors using minimalistic communication

Joshua Cherian Varughese[1,2], **Hannes Hornischer**[1], **Payam Zahadat**[1], **Ronald Thenius**[1], **Franz Wotawa**[2] **and Thomas Schmickl**[1]

[1] Department of Zoology, Institute of Biology, University of Graz, Graz, Austria
[2] Institute of Software Technology, Graz University of Technology, Graz, Austria

E-mail: ronald.thenius@uni-graz.at

Keywords: swarm intelligence, swarm robotics, swarm communication, swarm paradigm

Abstract

Numerous nature inspired algorithms have been suggested to enable robotic swarms, mobile sensor networks and other multi-agent systems to exhibit various self-organized behaviors. Swarm intelligence and swarm robotics research have been underway for a few decades and have produced many such algorithms based on natural self-organizing systems. While a large body of research exists for variations and modifications in swarm intelligence algorithms, there have been few attempts to unify the underlying agent level design of these widely varying behaviors. In this work, a design paradigm for a swarm of agents is presented which can exhibit a wide range of collective behaviors at swarm level while using minimalistic single-bit communication at the agent level. The communication in the proposed paradigm is based on waves of 'ping'-signals inspired by strategies for communication and self organization of slime mold (*Dictyostelium discoideum*) and fireflies (*lampyridae*). The unification of common collective behaviors through this Wave Oriented Swarm Paradigm (WOSP) enables the control of swarms with minimalistic communication and yet allowing the emergence of diverse complex behaviors. It is demonstrated both in simulation and using real robotic experiments that even a single-bit communication channel between agents suffices for the design of a substantial set of behaviors. Ultimately, the reader will be enabled to combine different behaviours based on the paradigm to develop a control scheme for individual swarms.

1. Introduction

Nature teems with various kinds of life forms with varying individual capabilities. Many of these lifeforms have been found to depend not only on individual capabilities but also on emergent group dynamics. Eberhart *et al* (2001) and Pulliam (1973) explains how swarming permits hunting together, foraging more efficiently, sharing food and collaborative defense against predators to increase their collective probability of survival and reproduction. The part each individual has to play in such emergent behaviors is often simple, yet the result that emerges on a group level is resilient to failures of individuals or other disruptive events (Middleton and Latty 2016). The simplicity and decentralized nature of the individual's behaviors producing resilient collective phenomena have attracted interest from engineers seeking to inherit properties such as simplicity, resilience, scalability etc in artificial systems. Numerous algorithms inspired by nature have been developed which enables simple agents to accomplish complex tasks such as optima finding (Schmickl *et al* 2008, Kennedy and Eberhart 1995), synchronization (Christensen *et al* 2009), flocking (Turgut *et al* 2008), source localization (Hayes *et al* 2003), task allocation (Labella *et al* 2006, Zahadat *et al* 2015) etc (Kennedy and Eberhart 1995) introduced an optimization algorithm, 'particle swarm optimization', inspired by flocks of birds. Schmickl *et al* (2008) introduced BEECLUST, an optima finding algorithm, inspired by a swarm of newly hatched bees. Groß *et al* (2006) and Kube and Bonabeau (2000) proposed swarm intelligent algorithms that enable robots with very limited individual abilities to transport large objects in a collective manner. A distributed algorithm for localizing the source of an odor in an environment is proposed and tested by Hayes *et al* (2003) on a swarm of robots.

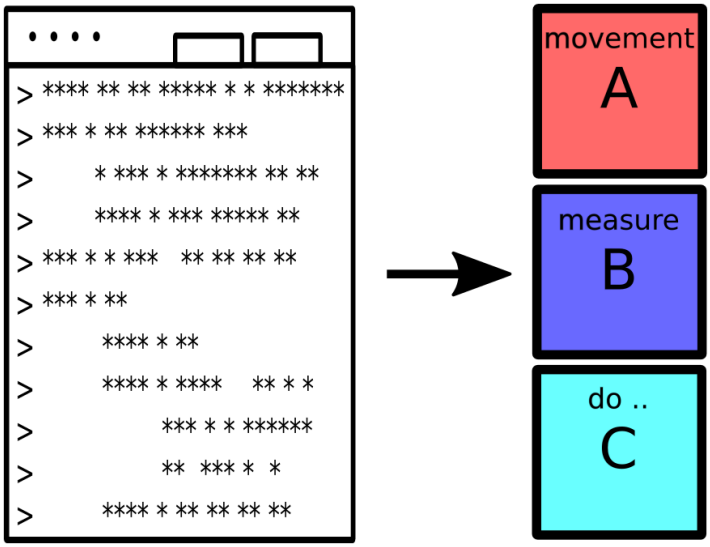


Figure 1. Schematic illustration of a meta control scheme. Every agent in WOSP follows a fixed basic operational structure. Behavioral changes are introduced not by major structural changes but by using the same communication structure and changing the action taken when agent receives an incoming communication. This structure allows the coherent design of fundamental behaviors or 'primitives' and thus the development of a meta control interface for controlling the swarm.

Although, many such algorithms have been suggested to accomplish various swarm level behaviors, hardly any underlying design similarities have been identified to unify these behaviors. One could explain the absence of such a unification by the possibly limiting nature of such an underlying design structure. Sumpter (2005) and Reid and Latty (2016) are two of the few papers that explicitly mention unification of swarm behaviors on a behavioral level. However, both of these papers are limited to the observation of similarities on a high level without actually suggesting a framework (Sumpter 2005) observed that although researchers collectively see vague similarities between different self-organizing behaviors, there is no universal theory or method so far that enables the exhibition of all or most of the collective behavior seen in biological systems. In Reid and Latty (2016), the authors addresses the question whether the collectives of slime mold fundamentally differ from other animal collectives. Furthermore, Reid and Latty (2016) demonstrated from existing research that collectives of very simple organisms like slime mold exhibit the principles of collective behavior outlined by Sumpter (2005).

Generally in contrast to existing approaches, the wave oriented swarm paradigm (WOSP) presented in this work, allows a swarm to inherit rich self organized collective behaviors thus unifying several common collective behaviors. As the name suggests, the paradigm enables a swarm to produce and propagate wave like communication signals and subsequently, use the characteristics of these communication waves to produce complex swarm level behaviors. Notably, one-bit communication suffices for the presented behaviors instead of complex messages and ultimately allows the design of both top-down control interfaces and autonomous swarms. The development of WOSP is closely associated with the subCULTron (subCULTron 2015) which deployed a heterogeneous swarm of underwater robots, with limited communication abilities (Thenius *et al* 2016), to monitor environmental parameters in the lagoon of Venice. The focus on minimalistic communication of WOSP is motivated by its association with subCULTron. WOSP provides a rich variety of self organized behaviors to the subCULTron swarm even though only simple local communication is available on the robotic platforms. Thenius *et al* (2018) explored the underlying concept of WOSP and presented some possible behaviors. In this paper, additional to what is presented by Thenius *et al* (2018), a large set of swarm behaviors is presented in simulation and some of the behaviors are validated using real robotic experiments. In addition, extensive review of related research is conducted in order to place the presented paradigm into perspective of existing work. For all presented behaviors the detailed structure and its design is described giving the reader an intuitive understanding of how to develop and design basic behavioral building blocks, which will be referred to as 'primitives'. Primitives serve as a basis for a meta control scheme and can be combined in various ways in order to produce complex collective behaviors as shown in figure 1. Primitives of WOSP function like behavioral units and policies in behavior-based systems (Brooks 1986, Matarić and Michaud 2008) where these units are used to combine basic behaviors into more complex behaviors. By contrast to behavior-based systems where inputs from various sensors are used to design behavioral policies, the primitives presented in this paper are based on responding to single-bit incoming communication. The design paradigm

introduced in this paper paves the way for programming swarms as shown in figure 1. Using such a meta control scheme, a programmer would be able to control a group of robots to execute a chain of primitives based on received pings. In order to enable the reader to effectively utilize the paradigm, a discussion regarding selection of parameters is also presented in this paper. It is worth noting that the exact syntax and grammar of a prospective language based on the presented paradigm is beyond the scope of this paper. Through this paper, the reader will be enabled to unify individual swarm behaviours commonly found in literature through a single communication paradigm. The unification of swarm behaviours enables programmers to design more primitives and further construct a meta control scheme for swarms.

In section 2, a review of relevant research is presented. Subsequently, in section 3 the bioinspiration for the communication mechanism from slime mold and fireflies is presented. In section 4, the communication mechanism as well as the fundamental concept of WOSP is introduced. In section 5 a set of primitives is introduced which are classified into three categories: internal organization, swarm awareness and locomotion. In section 6 possible methods for combining primitives and resulting complex behaviors are presented. In section 7, a discussion on parameters selection is performed to understand how parameter selection affects the performance of the primitives. In section 8, robotic experiments based on selected primitives are presented to validate the paradigm.

2. Related research

Research areas like 'human swarm interfaces' (Le Goc *et al* 2016) and 'swarm control' have some elements of unification implicitly involved due to the fact that a swarm of agents has to be able to execute multiple behaviors.

Le Goc *et al* (2016) employs a classical approach to controlling a swarm where users can interact with and control a swarm of robots, referred to as 'zooids', using gestures. The authors achieve a responsive swarm using an external projector for tracking the robots' position, assigning a goal position for each individual robot and then utilizing classical motion control strategies like proportional-integral-derivative (PID) control to move the zooids precisely from the start position to the goal position. Whereas the programmer has control of the precise movements of the swarms' members, the presence of a higher organizational entity is inherently necessary and thus depicts a classical example of top down control. In the pioneering work of Craig Reynolds, he introduced self propelled particles known as boids (Reynolds 1999) which exhibit self organized flocking and collision avoidance. Reynolds' boids are able to mimic a flock of birds whose individuals follow simple behavioral rules. Due to its simplicity and decentralized structure it is applicable to large swarms. Its focus is on the generation of realistic behaviors as found in natural flocks and hence limited in its versatility.

O'Keeffe *et al* (2017) presents a concept of self propelled particles with internal oscillators, or 'swarmallators'. Attractive and repulsive forces are then used in relation to the relative phase shift of the oscillators for generating a range of collective behaviors. The internal processes and states of the swarms' entities substantially influence and determine the interplay between individuals producing a small set of collective phenomena.

Nagpal (2002) introduces an algorithm for self assembly of identical agents on a surface into a predetermined global shape. Multiple gradients are developed by propagating messages starting from the agents at the edges in order to develop a relative positioning system among the agents. Subsequently, various shapes can be generated by manipulating the behavior of agents with particular gradient values. More complex shapes are achieved by repeating the process of generating gradients and folding along the specific areas of interest. Rubenstein *et al* (2014) uses a variation of the aforementioned algorithm to assemble various shapes in a self organized manner in a thousand robot swarm. Abelson *et al* (2000) unifies programmable self assembly and other similar research done by various researchers as 'amorphous computing' (Abelson *et al* 2000) uses a wave propagation mechanism as the agent to agent communication mechanism where certain 'seed' agents initiate a message to their neighbours. The neighbours then continue to pass on that message to their respective neighbours thus propagating waves of messages through the swarm. Each message contains the hop count and other relevant information depending on the task to be accomplished.

Pinciroli *et al* (2016) introduces the buzz programming language for programming swarms. The authors offer abstractions using domain specific language in order to enable the programmer to interact with a swarm at a high level rather than reinvent the wheel every time a swarm behaviour is developed. Buzz combines both bottom up and top down approaches to programming swarms and therefore presents a flexible programming language for swarm programming. Buzz is a framework with which a group of robots can each run a Buzz Virtual Machine, communicate and make decisions based on the shared information. Most of the common swarm behaviours can be implemented individually using the Buzz programming language. Buzz, similar to WOSP, relies on situated communication as a basic communication mechanism.

Among existing work, the paradigm presented in this paper exhibits most parallel characteristics with Buzz (Pinciroli *et al* 2016) and the approaches pre-

(a) Image from Čejková (2015) based on Ševčíková *et al* (2010)

(b) Image by Hristo Svinarov, `www.slowlight.bg`.

Figure 2. (a) A screenshot from a youtube video of *dictyostelium discoideum* aggregating into a slug. (b) A photograph of fireflies lighting up in unison.

sented under amorphous computing (Abelson *et al* 2000). However there are several points where WOSP significantly differs from both Buzz and amorphous computing.

In contrast to Buzz, WOSP is a design paradigm derived from a bio inspired minimalistic communication behaviour which unifies several swarm behaviours. Since Pinciroli *et al* (2016) has designed buzz language for swarms in general attending to the nuances and pitfalls of swarm programming, one could say that WOSP can be implemented in robots using the Buzz programming language. Therefore, Buzz is a full fledged programming language while WOSP is a design paradigm unifying several behaviours using minimalistic communication.

In contrast to Amorphous computing, which relies on multi-bit signals encoding information, the presented paradigm produces rich behavioral diversity with single bit communication. Another key difference between WOSP and the approaches presented under amorphous computing (Nagpal *et al* 2003) is that the presented paradigm refrains from using seed agents. Amorphous computing uses a hop count from the seed agent which enables the internal positioning of agents with respect to the seed agent which in turn allows a group of robots to organize themselves globally. In WOSP the agents initiate a single bit communication with the rest of the swarm randomly and in a decentralized manner. Interestingly, amorphous computing includes approaches which employ a wave propagation mechanism for swarm wide communication. Wave propagation used in amorphous computing is not periodic and differs substantially in implementation from the bio-inspired wave propagation mechanism used by WOSP as explained in section 3. Amorphous computing goes on to use the wave propagation for pattern formation by using languages such as origami shape language (OSL) (Nagpal 2002), growing point language (GPL) (Coore 2004). In contrast, the proposed paradigm will be used to program a swarm to unify a wide range of collective behaviors and ultimately enable programmers to combine basic behaviours.

3. Bioinspiration

The paradigm presented in this paper takes inspiration from the communication mechanisms used by slime mold and fireflies. In the following the relevant aspects of these lifeforms are briefly discussed.

3.1. Slime mold

Slime mold (*dictyostelium discoideum*), is a free living diploid life form which takes advantage of swarming behavior to survive in challenging environments. It has widely varying cooperation behaviors with other cells during its life cycle depending on the environment. When there are ample food sources, cells grow and divide individually without cooperation. In case of scarcity of food or other threats, significant cooperation between the cells begin. During its cooperative phases, the individual cells aggregate to form a multicellular organism and the collective begins to act similar to a single organism. The paradigm presented in this paper will chiefly consider the signaling behavior of slime mold during the aggregation of slime mold cells. For aggregating, some cells (centers) release Cyclic Adenosine Monophosphate (cAMP) into the environment to recruit surrounding cells to join the aggregate as captured in figure 2(a). This signal induces a short-lived chemical concentration spike around the recruiting cells (Siegert and Weijer 1992). Other slime mold cells that perceive the chemical signal will produce the emit the chemical themselves to relay the signal. Since all slime mold cells relay the chemical signals they receive, the original signal produced by the recruiting center rapidly travels through the swarm. Additionally, each cell needs around 12–15 s (Alcantara and Monk 1974) in between two cAMP signals. During this insensitive period, individual cells are insensitive

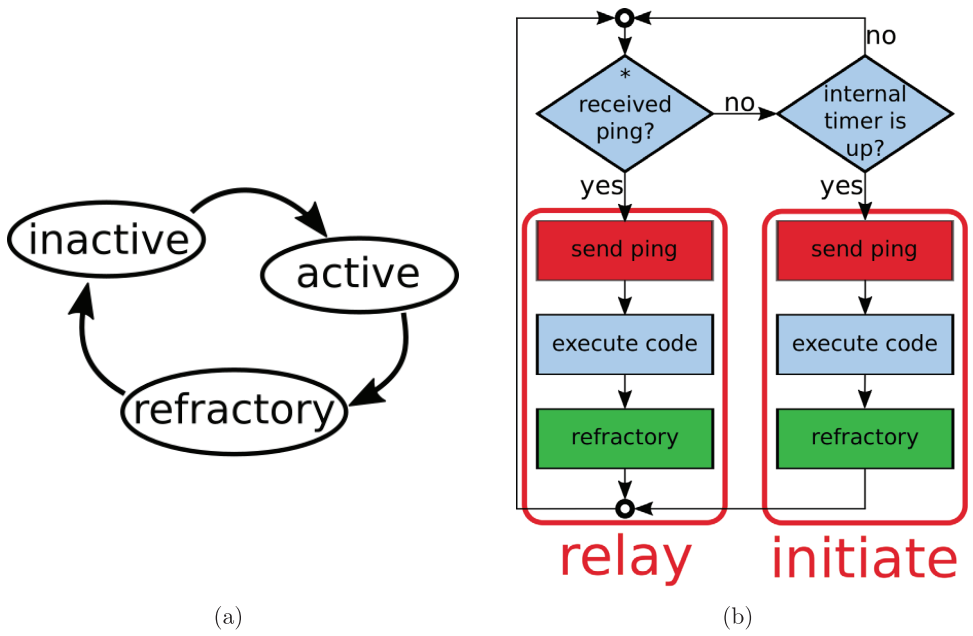


Figure 3. (a): three states of agents in WOSP: through external trigger (for example incoming ping) or internal trigger (timer), the agent transitions from the inactive state into the active state, where it sends a ping and optionally performs an action. Afterwards, it enters the refractory state, being insensitive to incoming signals until transitioning into the inactive state again. (b): the conceptual operating structure of an individual agent. If an agent in the inactive state receives a ping, it relays the signal by entering the active state and sending a ping itself. Depending on the primitive, a certain piece of code will be executed and then the agent enters the refractory state. If an agent's internal time is up it will initiate a ping following the same structure, however, executing a different code (specific to the primitive) in the active state.

to any cAMP pulses. This intermediate insensitive time acts as a 'refractory' period that prevents any echoing between two amoeba cells. The signal relaying mechanism described above forms the basis for spatio-temporal patterns known as scroll waves. Since the origin of the waves are at the recruiting cells, the amoeba can move towards the incoming signal to reach the recruiting center (Siegert and Weijer 1992).

3.2. Fireflies

Fireflies (*lampyridae*) are a family of insects that are capable of producing bio-luminescence to attract a mate or a prey (Buck and Buck 1966). Bio-luminescence of various families of fireflies has been a subject of elaborate study in the past (Buck and Buck 1966). Apart from being able to blink, fireflies are known to behave in cooperation to blink synchronously with other fireflies in order to attract mates or prey (Buck and Buck 1966). The synchronicity is a result of a simple mechanism by which initially the individual fireflies blink periodically. When a firefly perceives a blink in its surrounding, it blinks again and then resets its own blinking frequency to match the received blink (Camazine *et al* 2001). This is analogous to a phase coupled oscillator which adjusts its phase to match it to that of the faster one in the vicinity. This trait emerges into a quasi synchronized blinking pattern while the frequency of blinking will be influenced by the fastest blinking insect. An image of this synchronized blinking of fireflies is shown in figure 2(b).

4. WOSP—wave oriented swarm paradigm

The wave oriented swarm paradigm WOSP is inspired by the two aforementioned organisms: slime mold and fireflies. In particular, communication within the paradigm is based on cAMP waves propagating through a swarm of slime mold. Every agent within the considered swarm has the ability to send and receive single-bit (1-bit) information signals, which are henceforth referred to as 'pings'. All agents can enter three different states: an 'inactive' state, in which agents are receptive to incoming signals (responsive to cAMP or pings), an 'active' state, where they send or relay a signal (release cAMP or send ping) and optionally perform an action, which is followed by a 'refractory' state, where agents are temporarily insensitive to incoming signals. This is schematically shown in figure 3(a). Additionally, every agent has an internal timer, which counts down constantly. When it runs out, the agent enters the active state where it broadcasts a ping, thus initiating a ping wave through the swarm. An agent switches from the inactive state to the active state either due to its internal timer running out or due to it receiving a ping from one of its neighbors. The ability of fireflies to adjust and reset their individual frequency of 'blinking' is the inspiration for the concept of internal timers in this paradigm. This timer is reset right after running out, causing an agent to ping.

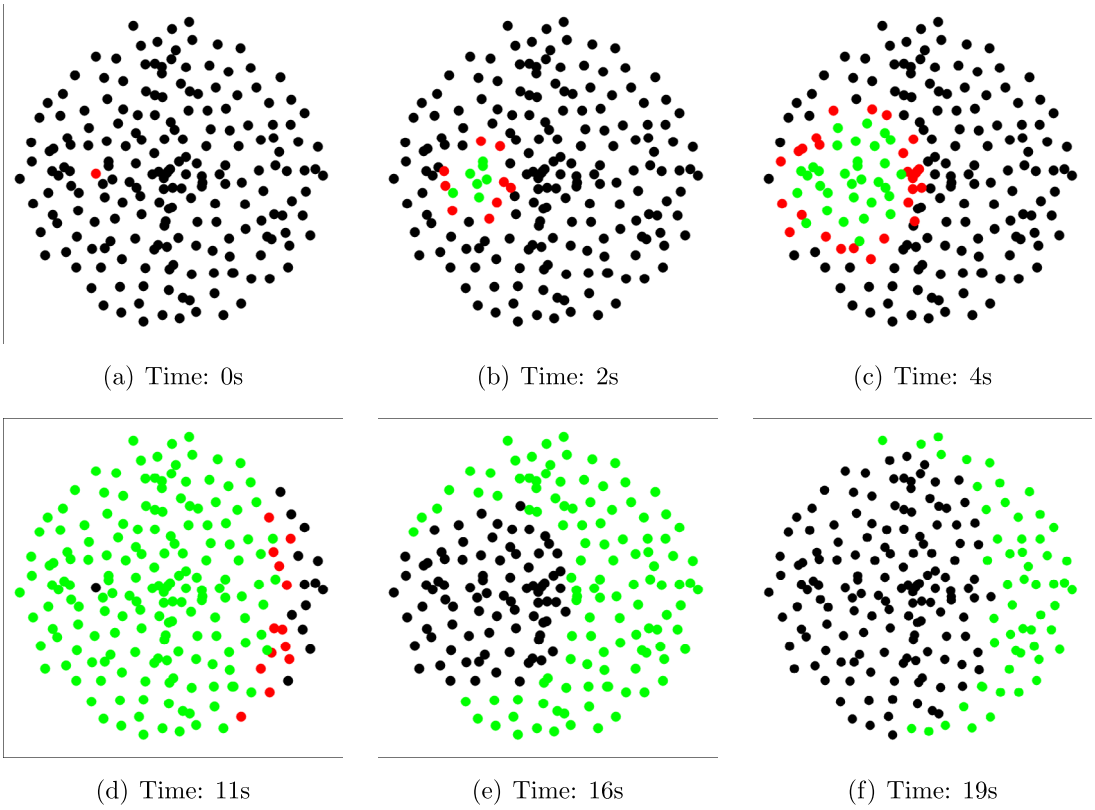


Figure 4. Illustration of wave based communication. In (a), almost all agents are in the inactive state, shown in black, except one agent which enters the active state and broadcasts a ping is shown in red. Afterwards it transitions into the refractory state, shown in green. Neighboring agents receive the signal and transition into the active state as shown in (b) and (c). The ping signal spreads in a wave like manner. In (d), the initiating agent transitions from the refractory state into the inactive state again. Due to a fixed duration of the refractory state, the transition into the inactive state as well spreads in a wave like manner, shown in (e) and (f). Parameters (as defined in more detail at the end of section 4): number of agents $N = 80$, physical size of the swarm in units perception range $R_s = 5$ r, refractory time in units timesteps $t_{ref} = 5$ s.

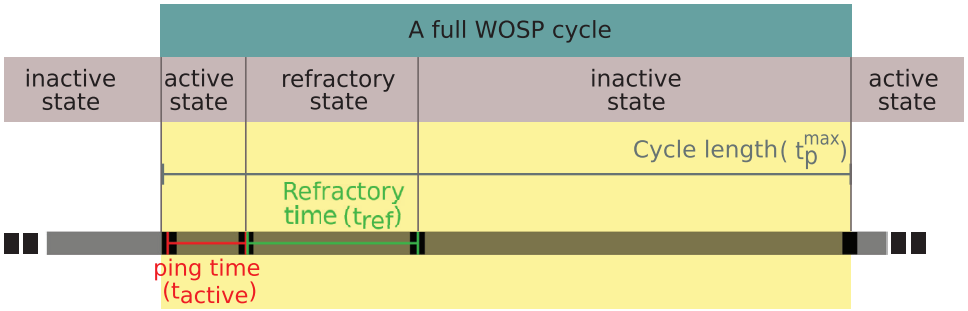


Figure 5. A schematic diagram of a full WOSP cycle including the timings involved and the names of the respective states.

This operational structure results in a wave like propagation of signals throughout the swarm. Agents in the inactive state get triggered to relay a signal, while the refractory state prevents the system from continuously signaling and thus, flooding the system. In figure 4, the propagation of waves is shown for a swarm of agents, each agent represented by a dot with the color denoting their state. The inactive state is denoted in black, the active state in red and the refractory state in green.

The aforementioned state transition constitutes the basic and fixed structure of an agent in WOSP and is shown as pseudo code in algorithm 1. This suffices for the behavior shown in figure 4 and is fixed for all agents. However, as will be presented later in this work, complex behavior can emerge when agents perform simple actions when relaying or initiating pings. This structure is conceptually shown in a flowchart in figure 3(b).

For the behaviors or primitives presented in this work, agents are not only able to send and receive pings, but also have a heading and a sense of directionality for incoming pings. The agents are assumed to be able to precisely perceive the direction from which the pings are received. In section 7, we will discuss the effects of this assumption on the performance of certain primitives for which angular resolution is relevant. Furthermore, for some tasks, agents have the ability to move in direction of their heading. For demonstration of the idea of WOSP, the agents are considered as point particles and thus, collision detection is ignored in this work.

Algorithm 1. Basic pseudo code for every individual agent within WOSP. Behavioral changes are only introduced by adding commands to the initiate- and relay-codeBlocks which are highlighted here. The timer t_p is initially set to a random value with upper limit t_p^{max}.

```
Data: Paradigm parameters
Result: -
state ← inactive;
timer(t_p) ← random integer ∈ (0, t_p^max];
While primitive do
    decrement timer(t_p);
    if agent in refractory state then
        wait for refractory_time;
        if refractory_time is over then
            state ← inactive
        end
    end
    if agent in active state then
        broadcast ping;
        state ← refractory
    end
    if agent in inactive state then
        listen for incoming pings;
        if ping received then
            state ← active;
            execute Relay-CodeBlock;
        end
    end
    if timer(t_p) ⩽ 0 then
        state ← active;
        execute Initiate-CodeBlock;
    end
end
Function Initiate-CodeBlock
    -
Function Relay-CodeBlock
    -
```

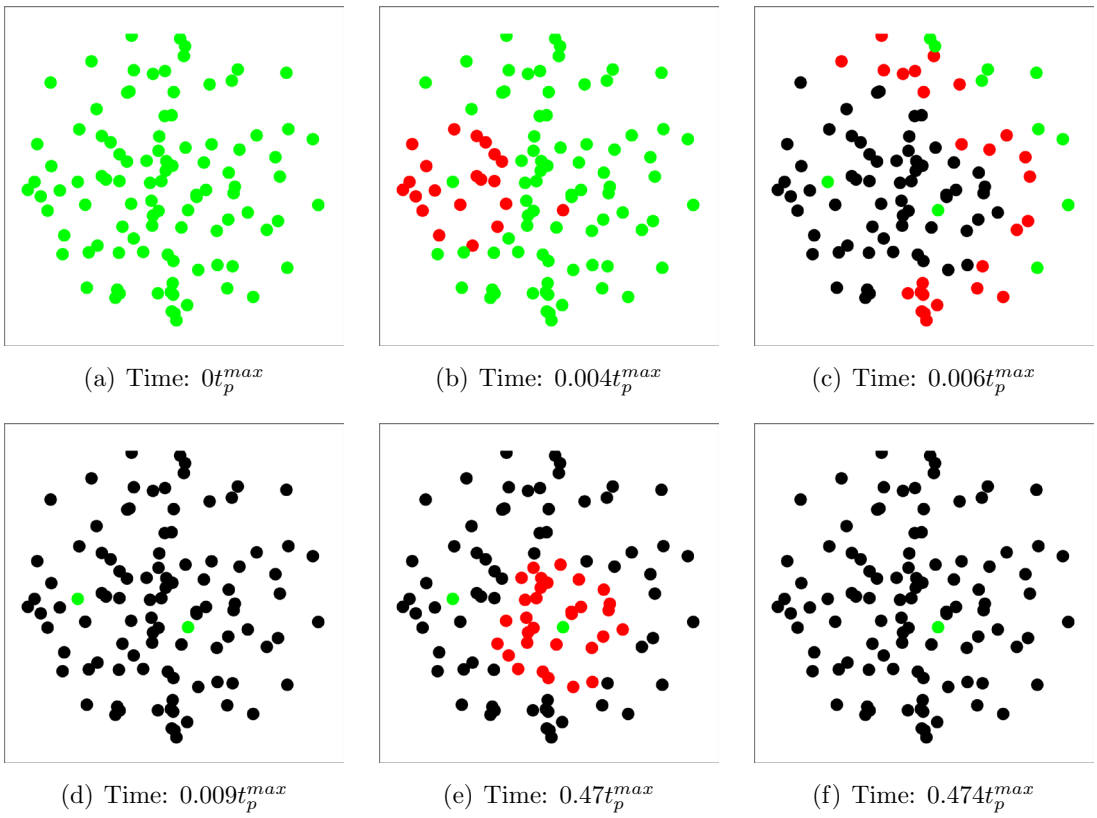
4.1. Parameters

The parameters and quantities used in this work are introduced and defined in this section. An analysis and discussion of the parameters is presented in section 7.

- Length of ping: in the numerical simulations presented here, time is measured in timesteps s. We consider a ping to be of length $t_{active} = 1$ s as shown in figure 5.
- Cycle length: every agent has an internal timer t_p which usually periodically resets to a maximum t_p^{max}. Therefore, the cycle length of each agent is t_p^{max}. For some primitives, however, the timer can be reset to a random number between $t_p \in (0, t_p^{max}]$.
- Number of swarm members: the number of agents constituting a swarm is defined as N. The minimum N necessary for the presented primitives to function is $N = 2$. As further elaborated in the discussion, the maximum can theoretically be arbitrarily large, limited by the speed of agent-to-agent communication and operational timescales.
- Refractory time: the refractory time t_{ref} denotes the time an agent remains in the refractory state, that is, insensitive to incoming pings. t_{ref} is set to a value as small as possible in order to maximize the time the agent is receptive to incoming pings, however needs to be set to a value sufficiently large so that a ping wave will not be relayed more than once.
- Perception range: for distances the basic unit is the perception range of agents r, the distance up to which an agent perceives the pinging of a nearby agent. For all simulation presented in this paper r is taken to be three space units, that is, $r = 3$ *units*.
- Length of steps: for primitives including locomotion, agents take discrete spatial steps within a timestep, where the length of their step d is set to one-sixth of a perception range $d = r/6$ if not stated otherwise. For decreasing d, the time until the tasks are completed increases, however, for too large values of d, the agents might move outside of the perception range with a single step and might thereby lose connectivity.
- Coverage: the physical spread of the swarm is defined as R_s and is given in units perception range r. If not stated otherwise, every agent in the swarm is initially randomly distributed within a circular area of radius R_s, such that every agent is connected to at least one neighbor.

5. Primitives

In this section, a set of primitives is presented, where small changes in the reactions of agents to incoming pings produce large scale complex behavior. For every primitive, both the plots of results as well as the code-block are presented. In order to ensure empirical validity, for each primitive presented in this section, ten independent simulations were successfully run if not stated or discussed otherwise. Additionally, section 7.2



(a) Time: $0t_p^{max}$ (b) Time: $0.004t_p^{max}$ (c) Time: $0.006t_p^{max}$

(d) Time: $0.009t_p^{max}$ (e) Time: $0.47t_p^{max}$ (f) Time: $0.474t_p^{max}$

Figure 6. Leader election in a swarm. Candidate leaders are shown in green, pinging agents in red and agents that do not consider themselves a candidate for leadership are shown in black. Initially, all agents consider themselves potential leaders as shown in (a). After receiving and relaying a ping, an agent will not consider itself a candidate leader anymore. Agents initiating pinging thus outrival agents around them. In (b)–(d) it is illustrated how several agents initiate pinging and not outrival each other (due to refractory time). Since only candidate leaders will initiate pinging, the remaining candidates then repeat the process as indicated in (e), until only a single candidate remains, as shown in (f). Parameters: $N = 100$, $R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

addresses empirical analysis and choice of parameters. The presented primitives are divided into three categories:

(i) Internal organization is about self-organization of the swarm on an internal level of each agent, including the primitives 'leader election', 'synchronization' and 'localize object'.
(ii) Swarm awareness includes the individuals' awareness about properties of the swarm or properties of itself within the swarm. The primitives 'localize swarm center', 'estimating number of swarm members' and 'estimate extremities of the swarm' are presented.
(iii) The category locomotion is about physically self-organizing or restructuring the swarm, including the primitives 'aggregation', 'moving collectively' and 'gas expansion'.

5.1. Internal organization: leader election

For various tasks it can be beneficial or even necessary for a swarm to have a certain agent 'leading' a swarm. Having a certain agent assigned to a special entity introduces the risk of a single point of failure, thus, disabling the entire swarm. Instead, the swarm can collectively elect a leader, thus reducing such a risk. In order to choose a leader, all agents initially consider themselves potential leaders, shown in figure 6(a) in green. An agent pinging is illustrated in red and an agent not considering itself a leader anymore is depicted in black. Every agent sets its timer to a random number within $t_p \in (0, t_p^{max}]$. As soon as an agent receives a ping before its own internal timer runs out it will not consider itself a candidate anymore and will also deactivate its internal timer. After an agent initiates a ping it will randomly choose a time t_p to initiate a ping again. This is shown as pseudo code in algorithm 2.

Algorithm 2. Code block for primitive 'leader election'.

Data: Paradigm parameters
Result: Leader election
.
.
Function *Initiate-CodeBlock*
    candidate ← true;
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    deactivate internal timer;
    candidate ← false;

Figures 6(b) and (c) show agents initiating ping waves and immediately outrivaling their surround-

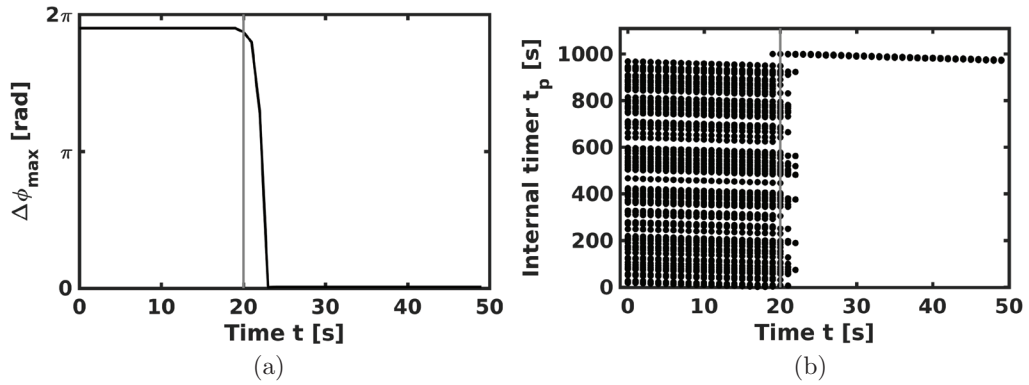


Figure 7. Analysis of the synchronization primitive. (a): onset of synchronized internal timers at $t \approx 20$ of a swarm of $N = 100$ agents, indicated with a gray vertical line. The order parameter $\Delta\phi_{max}$ is plotted against simulation time t. $\Delta\phi_{max}$ is calculated by determining the smallest phase interval containing the timers of all agents and then taking the maximum phase difference between two timers. After fully synchronizing at $t \approx 20$ s, the maximum phase difference decreased from $\Delta\phi_{max} \approx 5.5$ rad to $\Delta\phi_{max} \approx 0.04$ rad, which corresponds to a time interval of $\Delta t \approx 5$ s. This interval can be identified in (b), where the internal timers of the agents versus simulation time is shown. Every line of data points corresponds to the internal timer of a single agent, which incrementally counts down. All timers gradually decrease until at $t = 20$ s an agent's timer reaches $t_p = 0$ and thus initiates pinging. This marks the onset of the synchronization process and is marked with a gray vertical line. All agents relaying the ping then reset their timers. The reset signal propagates through the swarm and all agents reset and collectively count down in a quasi-synchronous manner. Parameters: $N = 100, R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

ing agents. The refractory mode prevents two initiating agents from outrivaling each other, however, more than one can be left as potential leader, as shown in figure 6(d). Since every remaining candidate again chooses a random time to ping, after few 'negotiation cycles' a single candidate which then can be considered the leader, will remain as shown in figures 6 (e) and (f). A proper selection of parameters is necessary to ensure that only a single leader remains after a certain number of negotiation cycles. The directives to select parameters are given in section 7. In order to empirically test the primitive, the primitive is said to have converged when there exists only a single candidate who considers itself the leader.

5.2. Internal organization: synchronization

It can be of great advantage for a swarm being able to perform coordinated actions. This primitive allows the swarm members to synchronize the sending of pings, allowing quasi-simultaneous coordination.

Every agent sets its internal counter to a random value between $t_p \in (0, t_p^{max}]$. If an agent receives a ping, it resets its internal counter to t_p^{max}. This is shown as pseudo code in algorithm 3. As a result, the first agent sending a ping (which is then being relayed and propagates wave-like through the system) resets the timers of all relaying agents to the maximum t_p^{max}. Hence, the entire swarm will ping quasi-simultaneously within a time period smaller or equal to the duration of a ping propagating from one end of the swarm to the other. In figure 7(a), the synchronization process for a swarm of $N = 50$ agents is shown via an order parameter $\Delta\phi_{max}$, which decreases with increasing synchronicity within the swarm. $\Delta\phi_{max}$ is calculated by determining the smallest phase interval containing the timer phases (ϕ_i) of all agents and then taking the maximum phase difference of all timer pairs as shown in the equations (1) and (2). At $t = 30$ s, the onset of synchronization is indicated with a gray vertical line. In figure 7(b), the corresponding internal timers of all agents are shown, incrementally decreasing with time. Every line of points represents the internal timer of one agent. At $t = 12$ s, an agent initiates pinging and thus resets the timers of all other agents such that at $t = 18$ s, all agents reset and are thus synchronized

$$\Delta\phi_{ij} = \mathbf{min}\{|\phi_i - \phi_j|,\ 2\pi - |\phi_i - \phi_j|\} \quad (1)$$

$$\Delta\phi_{max} = \mathbf{max}\{\Delta\phi_{ij} :\ i \neq j,\ i,j = 1,2\ldots N\}. \quad (2)$$

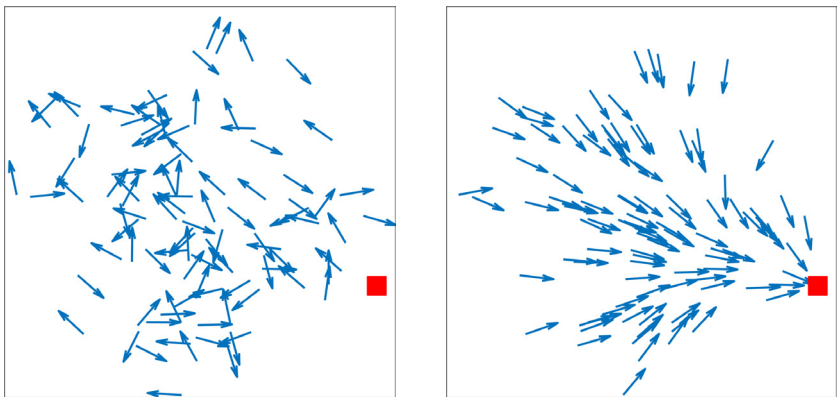
Synchronization primitive is said to have converged if $\Delta\phi_{max}$ is less than the phase difference corresponding to the time taken for a signal to propagate from one end of the swarm to the other.

Algorithm 3. Code block for primitive 'synchronization'.

Data: Paradigm parameters
Result: Synchronized Swarm
.
.
Function *Initiate-CodeBlock*
    timer(t_p)← t_p^{max};
Function *Relay-CodeBlock*
    timer(t_p)← t_p^{max};

5.3. Internal organization: localize object

For distributing information about spatial structure of the surrounding, a swarm needs to be able to communicate the location of nearby objects or events among its members. This primitive enables a swarm to collectively localize the direction of a direct path towards an object which one or few members of the swarm detect. Each agent refrains from initiating a ping unless it detects an object. Every agent receiving a ping, records the direction of the incoming ping. An estimate of the direction towards the object is then obtained by



(a) Time: $0\,t_p^{max}$ (b) Time: $0.49\,t_p^{max}$

Figure 8. The arrows representing the agents' estimation of the location of the object, the beginning of the arrow denotes the position of an agent. (a): the estimate is initialized with random direction at the start of the simulation. (b): the converged estimation of the location of the object after 0.49 t_p^{max}. All agents now point towards the object. Parameters: $N = 100$, $R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

taking a running average of the directions of incoming pings. The pseudo code is shown in algorithm 4.

Figure 8 shows the agents' estimate of the rough location of the object as an arrow placed at the position of the agent within the swarm. The red square represents an object which can only be detected by agents in its vicinity. Figure 8(a) shows the initial (random) orientation of the agents. With an increasing number of perceived pings, the estimate of direction towards the object increases in accuracy until agents accurately point towards the position of the object as shown in figure 8(b). For the purpose of empirically testing the primitive, the convergence is said to be reached when the average error between the individual agents' perception of the direction of the object and the actual direction of the object is below a particular threshold.

Algorithm 4. Code block for primitive 'localize object'.

Data: Paradigm parameters
Result: Agent knows rough direction of an object
.
.
Function *Initiate-CodeBlock*
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
    if no object detected **then**
        state ← inactive;
    end
Function *Relay-CodeBlock*
    record ping direction;
    current estimate ← average ping directions

5.4. Swarm awareness: localize swarm center

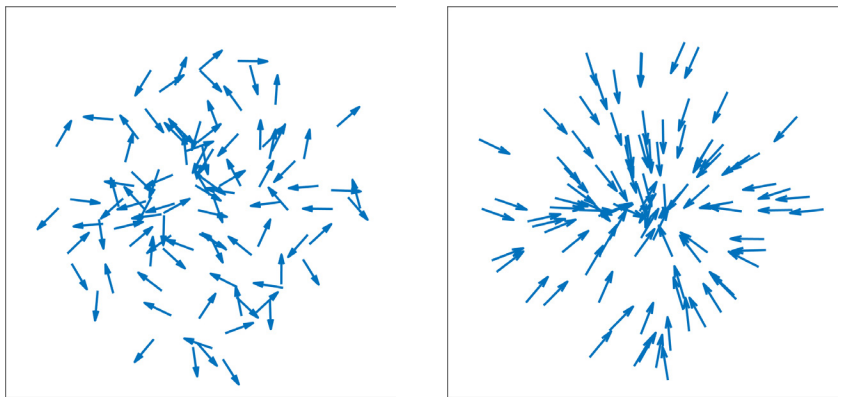
For a swarm to be able to execute spatially coordinated actions, the knowledge of the individual about the location of the center of the swarm can be of great advantage. This primitive enables each swarm member to identify the direction from where most signals originate from, which will be referred to as average origin of pings, or AOP. For a swarm of the presented type (circular, approximately homogeneously distributed, agents have several communication neighbors) this direction coincides with the direction towards the physical center of mass of the swarm. Each agent sets its internal counter t_p to a random value between $t_p \in (0, t_p^{max}]$ and as soon as a counter reaches $t_p = 0$ an agent sends a ping. When an agent receives a ping it stores the direction of the incoming ping and averages over all stored directions. This is shown as pseudo code in algorithm 5.

Algorithm 5. Code block for primitive 'localize swarm center'.

Data: Paradigm parameters
Result: Agent knows rough direction of swarm center
.
.
Function *Initiate-CodeBlock*
    center estimate ← mean of previous estimates;
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    record ping direction;
    current estimate ← average ping directions

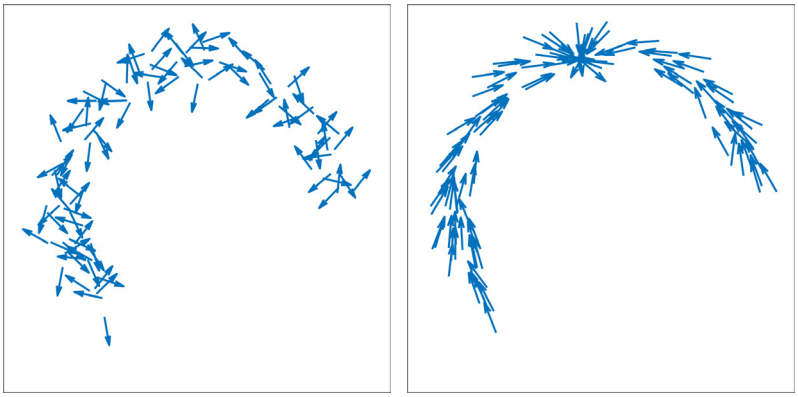
Figure 9 shows a swarm in its initial state and after it equilibrated where every agent's orientation is denoted by an arrow at the position of the agent in the swarm. Initially, the heading is random. After equilibrating, the agents on the outside accurately point towards the center of the swarm.

However, this primitive does not result in the detection of the geometrical center of the swarm for all spatial configurations of agents since agents locate the direction from which they receive most pings. This is illustrated in figure 10 where the swarm is shaped as a ring segment. Instead of agents pointing towards the center of the ring segment, they orient themselves towards neighboring agents, ultimately leading to the



(a) Time: $0\,t_p^{max}$ (b) Time: $0.38\,t_p^{max}$

Figure 9. Agents' estimation of the direction towards the center of the swarm. The beginning of an arrow denotes the position of an agent. (a) shows the initial estimates of each agent as arrow at its position in the swarm. (b) shows the converged estimates after $t = 0.38\,t_p^{max}$. Parameters: $N = 100, R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.



(a) Time: $0\,t_p^{max}$ (b) Time: $0.025\,t_p^{max}$

Figure 10. Agents' estimation of the direction towards the center of the swarm when it does not coincide with the geometrical center of the swarm. The beginning of an arrow denotes the position of an agent. (a) shows the initial estimates of each agent, (b) shows the converged estimates after $t = 0.025\,t_p^{max}$. Parameters: $N = 100$, radius of ring $R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

agents which would be in the center of the swarm if the swarm was stretched to a straight line.

When adequately selected, the t_p^{max} will be large enough for every agent to ping in a slot in which its ping wave does not collide with another wave, therefore providing a more accurate estimate of the average origin of pings. For empirical testing of the primitive, the primitive is considered to have converged when the average difference between the agents estimate of the direction of the center and the actual direction to the center is below a predefined threshold.

5.5. Swarm awareness: estimating number of swarm members

For some tasks a swarm may need to be constituted of a certain number of agents in order to effectively operate. For instance, a swarm may need to check if the number of its members has substantially changed due to loss of members or merging with another swarm. The paradigm presented here enables the swarm members to decentrally estimate the number of swarm members without needing an external observer. Each agent sets its internal counter t_p to a random value between $t_p \in (0, t_p^{max}]$. Whenever an internal timer is up, an agent will initiate pinging and randomly reset its timer to $t_p \in (0, t_p^{max}]$. Each time an agent receives a ping it relays the signal and increments a counter N_{count}.

Furthermore, every time an agent initiates pinging (when one internal cycle has passed), it will store its counter N_{count} as its estimate of the number of swarm members for the past cycle. The average of those estimates will be the agent's opinion of the number of members in the swarm N_{est}. This is shown as pseudo code in algorithm 6.

Algorithm 6. Code block for primitive 'estimating number of swarm members'.

Data: Paradigm parameters
Result: Agent knows approximate number of members in the swarm
.
.
Function *Initiate-CodeBlock*
    estimate(N_{est}) ← mean of previous swarm size counters;
    counter(N_{count}) ← 0;
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    increment counter N_{count};

In figure 11 the estimate N_{est} averaged over all members of a swarm versus simulation time is shown. The estimate quickly increases before slowly converging to $N_{est} \approx 34$. The error bars represent the standard deviation, thus indicating that the estimates of all agents are closely distributed around the mean. The estimate converges to a value significantly lower than the actual number of swarm members, however, for the same swarm the estimate consistently converges to the same (lower) estimate. The estimate converges to a lower value than the actual number of agents in the swarm because ping waves are initiated by more than one agent at roughly the same time, causing several ping waves to coincide. As a result, agents in the swarm detect only a single wave and accordingly increment the estimate only once. For larger cycle lengths, ping waves collide less and the estimate is closer to the actual number of agents. An empirical study on the dependence of N_{est} can be found in section 7. To empirically test the primitive, all runs in which the estimate of the number of agents reached 75% of N, that is $N_{est} \geqslant 0.75\ N$, were considered to be successful.

5.6. Swarm awareness: estimate extremities of the swarm

This primitive enables the agents in a swarm to determine the extremities of the swarm by identifying gaps in the direction of incoming pings. In order to do so, each agent sets its internal timer t_p to a random value between $t_p \in (0, t_p^{max}]$. As previously explained, this will result in agents pinging at random time slots and each agent relaying the received pings. The agents then bin each of the pings received into four directions of $\alpha = 90°$ each. If there is at least one empty bin with no pings received, then the agent perceives itself as being on the periphery of the swarm. Pseudo code is shown in algorithm 7.

Algorithm 7. Code block for primitive 'localize object'.

Data: Paradigm parameters
Result: Agent knows if it is at the periphery
.
.
Function *Initiate-CodeBlock*
    if *Is at least one bin empty?* **then**
        periphery ← true;
    else
        periphery ← false;
    end
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    record ping direction;
    bin incoming ping directions into bins of 90°;

Figure 12 shows the perception of agents regarding their position in the swarm. Initially, no agents perceive themselves to be at the periphery of the swarm, denoted by the black color of agents in figure 12(a). As agents receive more pings from the surrounding agents, they are able to estimate their own position more accurately within the swarm as shown in figure 12(b) where yellow colored agents perceive that they are at the periphery of the swarm.

To empirically test the primitive, a simulation run was considered successful if more than 80% of the agents in the periphery successfully identified themselves as being at the periphery based on the incoming pings.

5.7. Locomotion: aggregation

Considering a swarm of agents with the ability to move and spatially arrange itself, it needs to be able to gather or aggregate in order to regroup itself. In order to achieve that, every agent randomly sets its internal counter t_p to a random value between $t_p \in (0, t_p^{max}]$. An agent receiving a ping will relay it and then move a small distance towards the incoming ping. Gradually, all agents move towards each other. The pseudo code for aggregation is shown in algorithm 8.

Figure 13 shows a swarm aggregating according to algorithm 8. From its initial state the swarm steadily moves towards its average origin of pings, causing it to aggregate at the center of the swarm. Figure 13(d) shows the aggregated state of the agents as well as each agent's trajectory as blue line. This illustrates how agents tend to follow the paths of their fellow members of the swarm, producing a root-like trajectory structure. For illustrating the aggregation process, figure 14 shows in blue the average root mean square

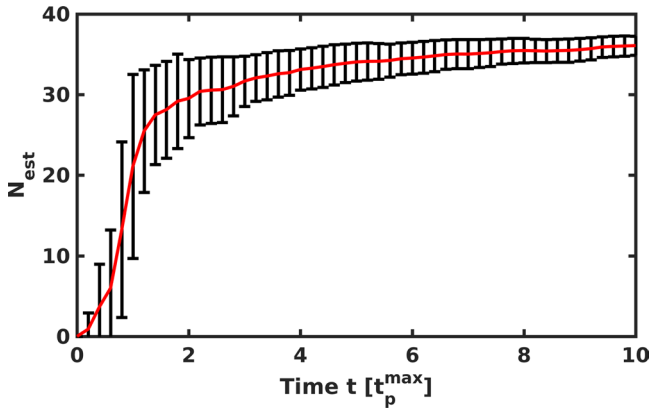
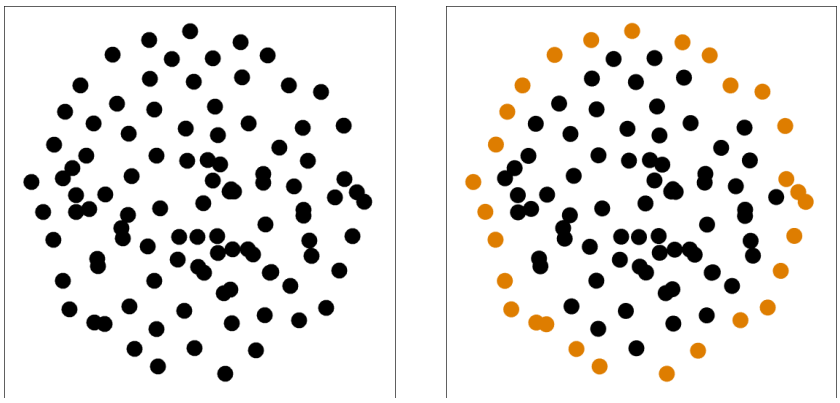


Figure 11. Estimated number of swarm members averaged over all agents in the swarm versus time. The error bars represent the standard deviation. The estimate steeply increases from $N_{est} = 0$ to $N_{est} = 30$ at $t = 3\, t_p^{max}$ before it gradually converges to its final estimate of $N_{est} \approx 35$. Parameters: $N = 50$, $R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 2500$ s. In order to put the timescales into perspective of cycle lengths, the time axis is measured as multiples of t_p^{max}.



(a) Time: $0\, t_p^{max}$ (b) Time: $1.4\, t_p^{max}$

Figure 12. Figures show the agents' perception of their location within the swarm. Yellow colored agents perceive that they are at the periphery and the black colored agents perceive themselves as not being at the periphery. (a) shows the initialization at the start of the simulation with all agents perceiving themselves as 'not being at the periphery'. (b) shows the converged perception of the agents after $1.4\, t_p^{max}$. Parameters: $N = 100$, $R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

distance (R_{rms}) of all agents from the center of the mass of the swarm which represents the spread of the swarm members. To empirically test the primitive, a simulation run was considered successful with the termination condition $R_{rms} < 0.3r$.

Algorithm 8. Code block for primitive 'aggregation'.

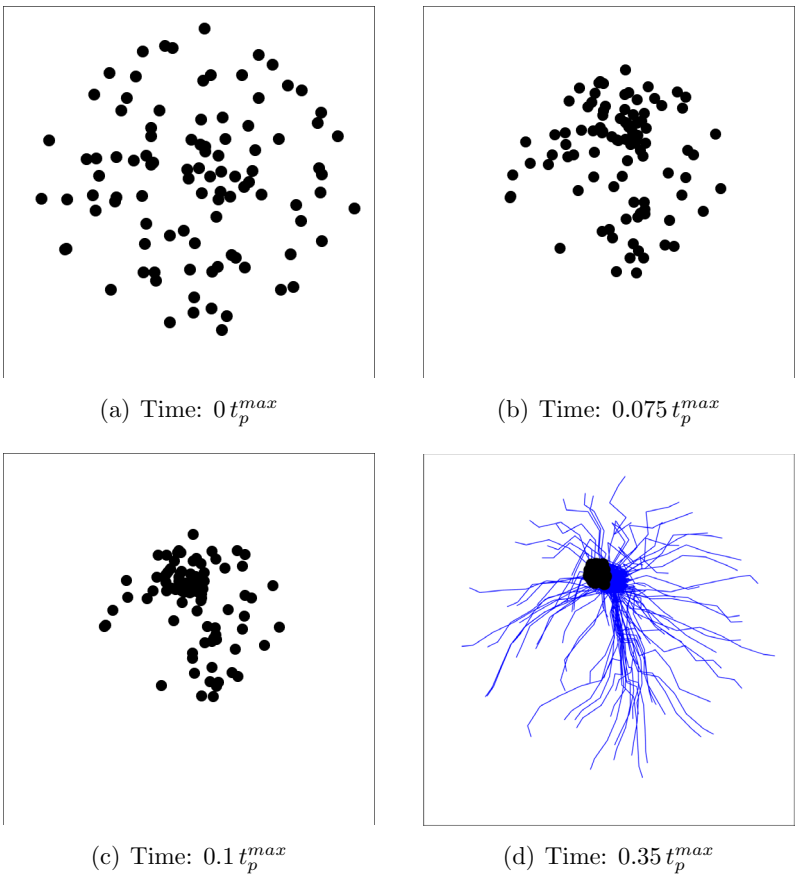
Data: Paradigm parameters
Result: Aggregated swarm
.
.
Function *Initiate-CodeBlock*
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    timer(t_p) ← t_p^{max};
    record ping direction;
    Calculate average of incoming pings;
    move towards incoming ping;

Considering that a swarm needs to aggregate at a specific location, the primitive can be changed in such a way that only certain agents, which for example perceive stimuli such as the presence of an object, are able to initiate pings. This is shown in figure 15.

The stimulus can also be an event or can be connected with a gradient. Considering agents with the ability to perceive light intensity, the agents will be able to aggregate at the brightest spot if every agent sets its internal counter to a value proportional to its perceived brightness. The agents at the brightest spots will statistically ping first. Furthermore, every agent receiving a ping will reset its counter, thus allowing the agents at the brightest spot to hijack the swarm. This process when repeatedly executed will result in a gradient taxis behavior as presented in Varughese *et al* (2016).

5.8. Locomotion: moving collectively

For the mobility of a swarm, the ability to collectively move to a specific location can be crucial. For letting the entire swarm move towards a certain direction, a single agent serves as leader. This leader exclusively initiates pings and gradually moves along a trajectory



(a) Time: $0\,t_p^{max}$ (b) Time: $0.075\,t_p^{max}$

(c) Time: $0.1\,t_p^{max}$ (d) Time: $0.35\,t_p^{max}$

Figure 13. Figures show aggregation of a swarm. Initial state of the swarm is shown in (a), in (b) and (c) it steadily aggregates. The final state and trajectories of each agent for the entire simulation are shown in (d) in blue lines. Parameters: $N = 100, R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

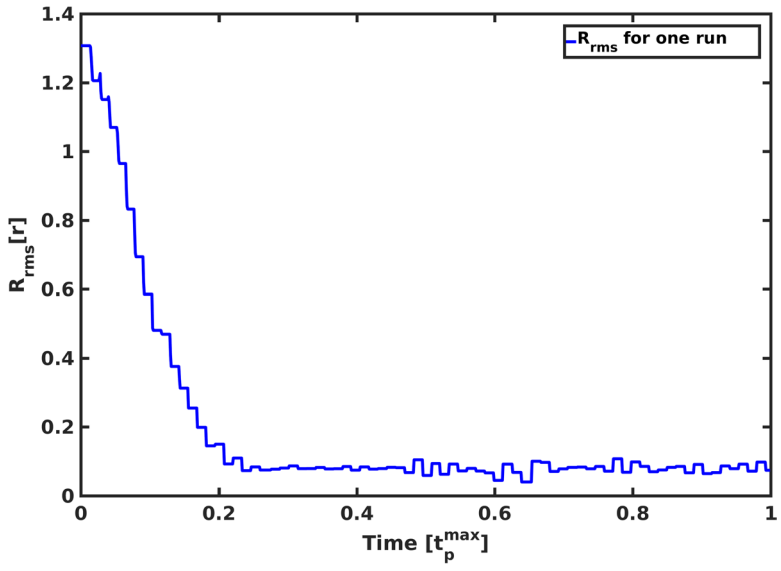


Figure 14. Average root mean square distance of all agents from the center of the swarm, R_{rms}, plotted against time. The blue line plot shows the R_{rms} of the swarm in the simulation shown in figure 13. The R_{rms} linearly decreases until $t_p^{max} \approx 1$ when the swarm has almost fully aggregated. The decrease of R_{rms} occurs in discrete steps, corresponding to ping waves causing all agents to move towards each other quasi-simultaneously by a step of a predefined length d. Parameters: $N = 100, R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

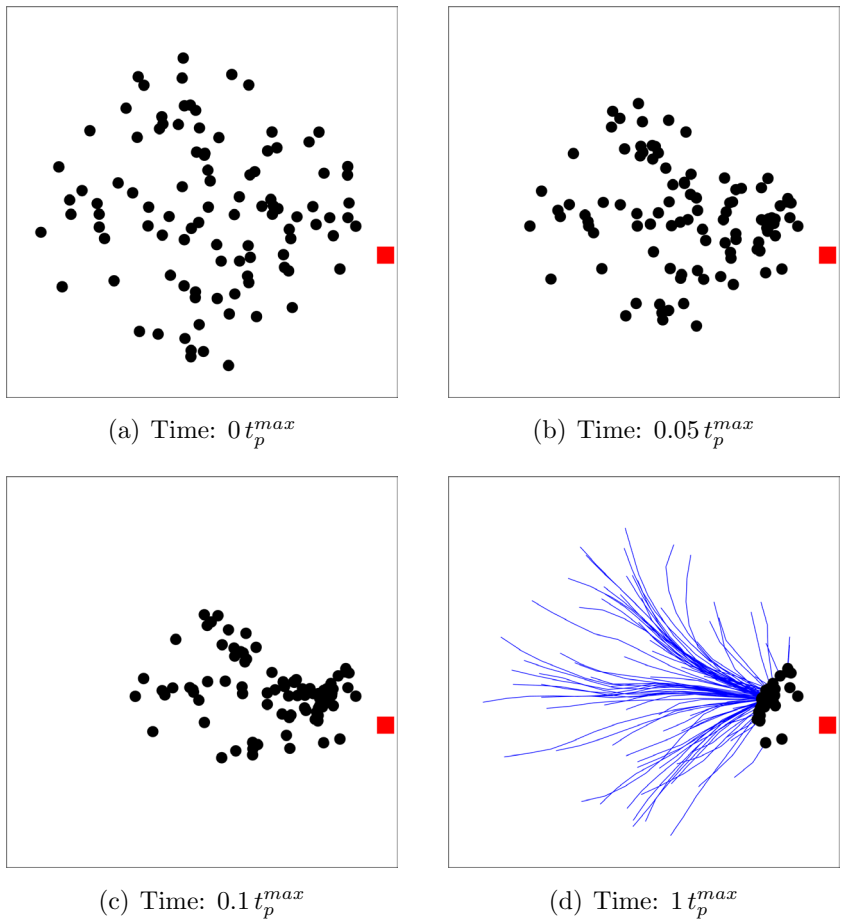


Figure 15. A swarm aggregating at an object, marked as red square on right hand side of the shown system. Initial state of the swarm is shown in (a). The swarm gradually aggregates at the object in (b) and (c) until every agent directly perceives the object in (d). The trajectories of agents for the entire simulation is shown in blue lines. Parameters: $N = 100, R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

leading to the target location. All agents receiving pings will move towards the direction of it and thus, follow the leader. The pseudo code is shown in algorithm 9.

Algorithm 9. Code block for primitive 'moving collectively'.

Data: Paradigm parameters
Result: Swarm follows a leader
.
.
Function *Initiate-CodeBlock*
    leader ← true;
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    deactivate timer;
    leader ← false;
    record ping direction;
    calculate average of incoming pings;
    move towards incoming ping;

Figure 16 shows a swarm aggregating towards a leader located at the far right end of the swarm, which steadily moves towards the right. While following the leader, the remaining swarm forms a line behind it, being lead away. This primitive can be viewed as 'aggregation at a specific, moving agent'. For choosing a leading agent, the primitive 'leader election', which was earlier introduced, can be executed prior to this primitive. This primitive is said to have converged, when the R_{rms} of the swarm is below a set threshold, depending upon the size of the swarm.

5.9. Locomotion: gas expansion

The primitive 'gas expansion' enables a swarm to uniformly expand. Each agent sets its internal counter t_p to a random value between $t_p \in (0, t_p^{max}]$. As soon as the internal counter reaches $t_p = 0$ an agent sends a ping. Each agent moves a small step away from the incoming pings. As soon as an agent does not receive pings anymore, it does not move any further. The agents can then reconnect with their swarm members by moving back, in the opposite direction of the previous step, or by integrating their entire trajectory and thus, finding their way back until they perceive signals again. Depending on the communication abilities of the swarm, the perception range or sensitivity can be temporarily decreased during the expansion such

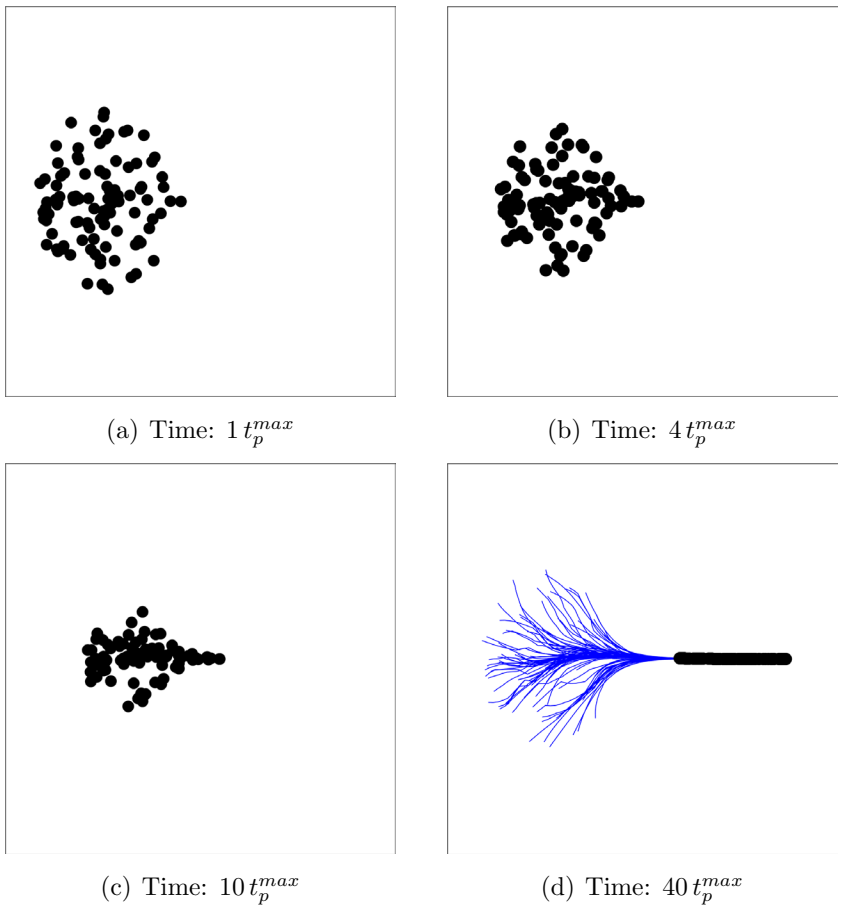


Figure 16. This figure shows a swarm being led by a single agent towards the right. From the initial state in (a), the swarm aggregates towards the leading agent in (b) and (c). In (d), the swarm forms a line following the leader. The trajectories of each agent for the entire simulation is shown in blue lines. Parameters: $N = 100$, $R_s = 2.33$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

that afterwards the agents will again be connected. See algorithm 10 for the pseudo code. In figure 17 (a) an initially densely packed swarm is shown, which then expands in figures 17(b) and (c) until it is fully expanded in figure 17(d). It has to be noted that since the gas expansion primitive relies on agents getting reconnected to the swarm, there is a chance of the agents getting lost especially in the case of real robots. Some strategies can be used to solve this problem and a discussion on connectivity can be found in section 9. To test the primitive empirically, a simulation run was considered successful if the average distance between any agent and its neighbors was greater than $0.8r$.

Algorithm 10. Code block for primitive 'gas expansion'.

Data: Paradigm parameters
Result: Expanded swarm
.
.
Function *Initiate-CodeBlock*
    timer(t_p) ← random integer ∈ $(0, t_p^{max}]$;
Function *Relay-CodeBlock*
    timer(t_p) ← t_p^{max};
    record ping direction;
    calculate average of incoming pings;
    move away from incoming ping;

6. Combining primitives

By combining primitives more elaborate behaviors can be produced and by enabling a swarm to switch between a set of primitives it can operate autonomously. The most intuitive way of combining primitives is to execute primitives one after another each for a fixed period of time as schematically shown in figure 18. This allows the design of complex tasks which can be executed by the swarm autonomously.

6.1. Combining primitives: exploration

An example for sequential execution of primitives producing an autonomously acting swarm is a collective exploration procedure, shown in figure 19. The following sequence of primitives is executed periodically: aggregation, leader election, moving collectively, gas expansion. In figures 19(a) and (b) the swarm aggregates and then determines a leader in figure 19(c)). This leader chooses a random direction and leads the swarm to a new location, as shown in figures 19(c) and (d). Then, the entire swarm expands before aggregating again and restarting the same procedure. Due to the limited abilities of the individual members of the swarm, they have no awareness of the collective state or if the execution of a primitive has

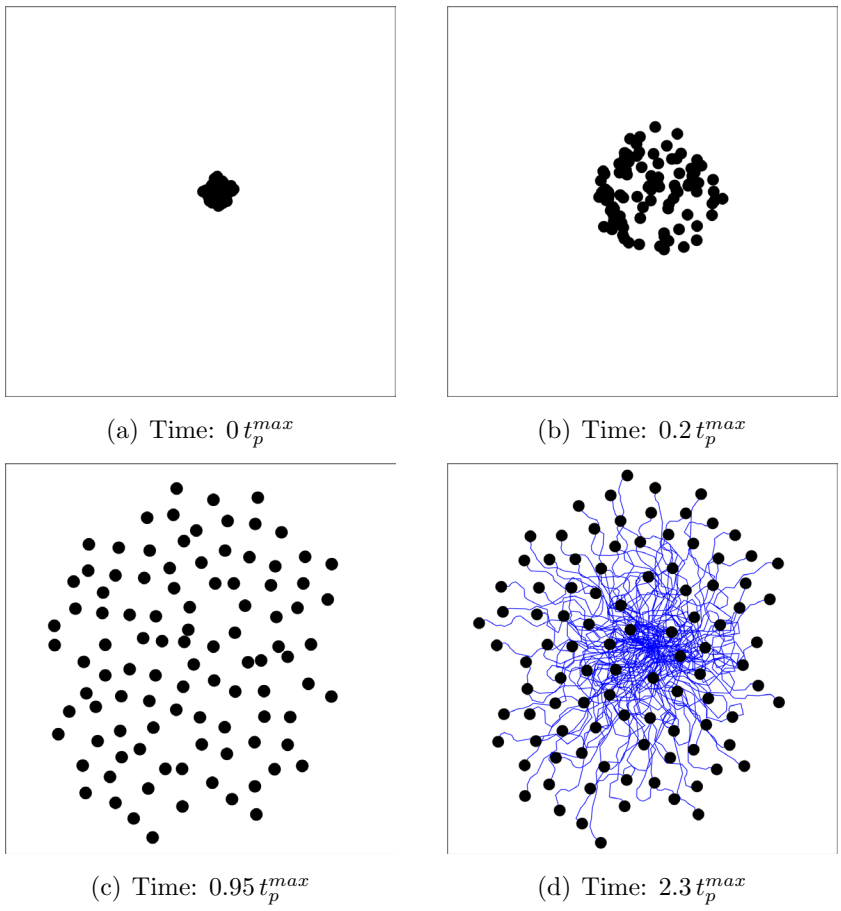


Figure 17. A swarm performing the primitive 'gas expansion'. The initial state of the swarm is shown in (a), where it is aggregated. In (b) and (c) it gradually expands. The final state is shown in (d) with the trajectories of each swarm depicted as blue lines. Parameters: $N = 100, R_s = 0.67$ r, $t_{ref} = 10$ s, $t_p^{max} = 1000$ s.

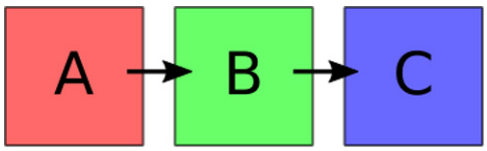


Figure 18. Schematic illustration of sequential execution of three primitives A, B and C.

converged. For the present example, the execution times of all primitives were fixed.

6.2. Combining primitives: collective transport

An instance of combining primitives in a sequential manner can be demonstrated by solving a simplified version of the collective transport problem. A scenario where a group of agents transports an object is presented here. In order to be able to move an object, additional physical abilities of agents are assumed. An agent can detect the presence of an object or a goal in its near vicinity (one space unit). Additionally, it is also assumed that the agents can exert a force on the object. As for the motion of the object, it is assumed that if more than three agents exert force on the object, it can be moved along with the movement of the respective agents. The sequence of primitives executed here are as follows: gas expansion, aggregate at goal (only by agents which detected the object). The magenta and grey patches in figure 20 represent the object to be moved and the goal, respectively. The swarm executes gas expansion in order to find both the goal and the object to be moved as shown in figure 20(b). Those agents that detect the goal generate pings and those agents which find the object attach to the object and move it towards the incoming pings as shown in figures 20(c)–(f). Through this combination of primitives, the agents can move the object to the goal without explicitly being told where the target is.

Evidently, the collective transport shown here is a simplified version. In order to present combining WOSP primitives to solve complex problems, complexities specifically associated with collective transport have been ignored. The area of collective transport has received much attention in the previous years. Broadly, the collective transport problem has been solved by either lifting/grasping (Groß *et al* 2006) the object or pushing/caging (Fink *et al* 2008, Kube and Bonabeau 2000) the object. Using the approach presented here, the position of the goal is implicitly associate with the direction of the pings. Therefore, the need for each agent to know the position of the goal (global knowledge) is obviated. The advantage of using pings to guide the 'pushers' to the goal comes at the cost of needing a large number of agents. The number of

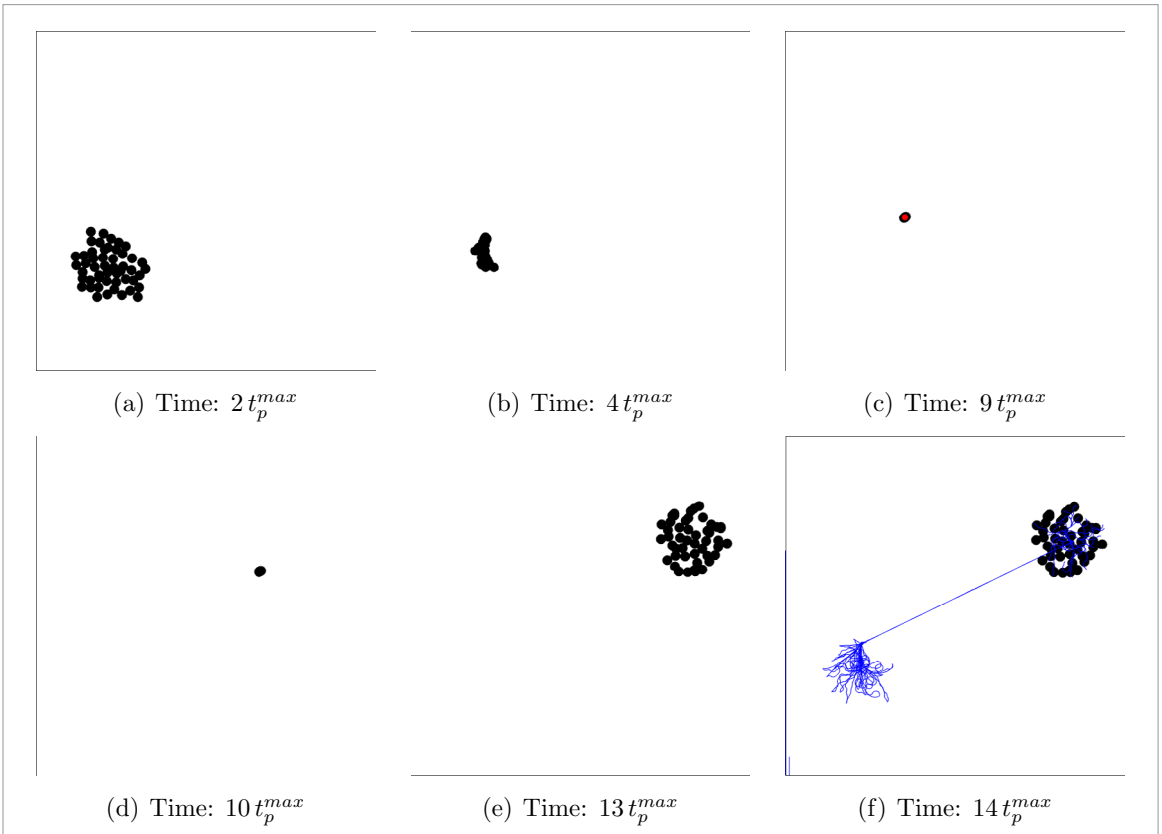


Figure 19. Consecutive execution of the primitives aggregation, leader election, moving collectively and gas expansion as example for an exploring routine of an autonomous swarm. The swarm prepares for changing its location and thus aggregates from (a) to (c). It then decides upon a leading agent (marked in red) which then leads the swarm towards the top right of the system, a target area, shown in (c) and (d). The swarm expands again as shown in (e). In (f) the final state is shown along with the trajectories of all agents over the entire simulation. Parameters: $N = 50$, t_{ref} and t_p^{max} vary for each primitive.

agents required for this task will depend on the sensor range of the agents. A large enough swarm with a specific sensor range can expand and discover both the source and the goal without losing connectivity.

It is important to note that the aim of presenting this scenario is not to present a perfect solution for the collective transport problem but rather to point out the wide range of possibilities using a minimalisitc communication paradigm. Using WOSP, a swarm programmer can combine several basic primitives with the physical abilities of the agents in order to accomplish complex behaviors.

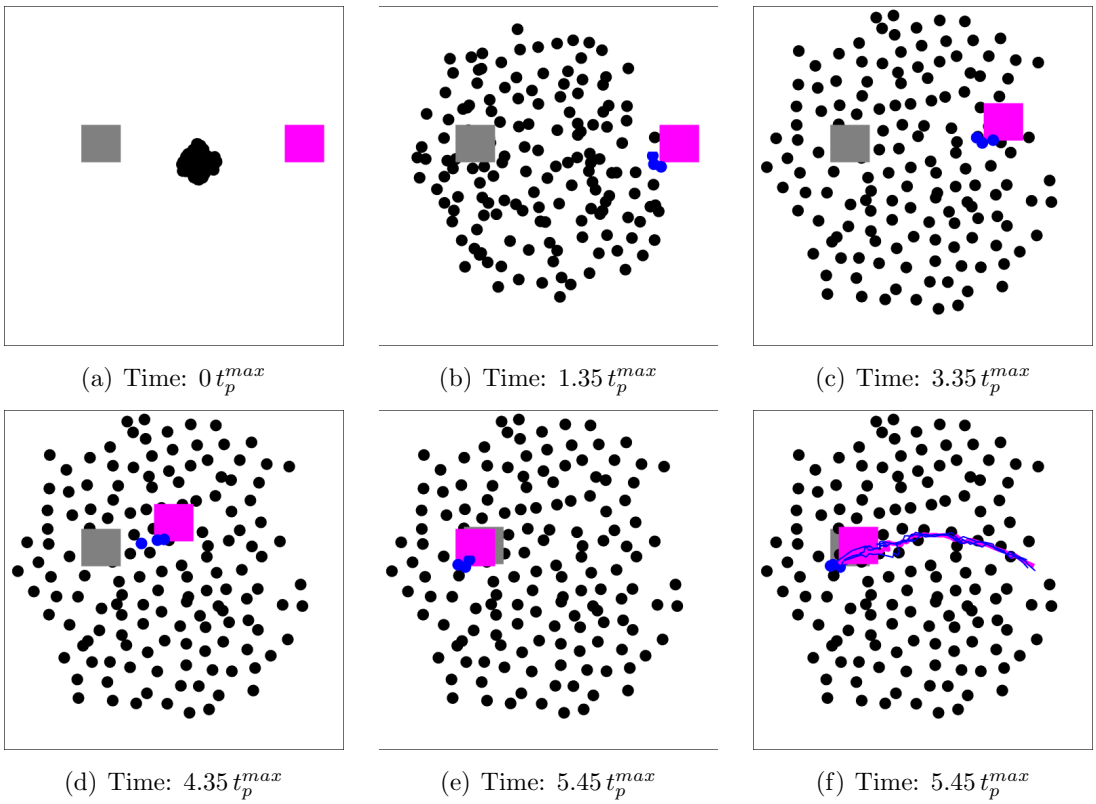
Another approach to combining primitives is to execute several primitives in an interleaved manner. Such a combination allows the emergence of a larger variety of complex behaviors as done by combining behaviors in behavior-based systems (Maes 1989, Matarić 1997). For executing several primitives in a quasi-simultaneous manner, the previously presented single-bit communication must be extended. A simple option is to introduce several individual layers of single-bit communication, one for each primitive. Alternatively, multi-bit signals could be used to encode each waves of each primitive separately. However, interleaved execution of primitives is beyond the scope of this paper and therefore, will not be presented. Additional communication payload could also be used for other purposes for exchanging information regarding the completion of primitives. In the interest of keeping communication single-bit, a time based sequential execution of primitives is employed in this section. In many practical applications, such a time based execution of primitives might be too limiting to produce meaningful combination of basic behaviors. With more sophisticated sensing mechanisms or by introducing a global observer as seen in Le Goc *et al* (2016), a swarm can acquire the ability to understand if the execution of a primitive is completed. For example, if the intensity of incoming communication signal is available, the agents can estimate how close the neighboring robot is and this in turn, could be used to estimate the state of 'aggregation' of the swarm. It can therefore be stated that while WOSP offers a minimalistic paradigm for design of swarm behaviors, various additions can be made to make the swarm more versatile.

7. Analysis and discussion of parameters

In this section, the influence of parameters and initial conditions of the swarm on the functionality of primitives are examined. Exemplary scenarios are chosen, which allow to infer the relation between parameters and the swarm's behavior.

7.1. Parameter dependencies

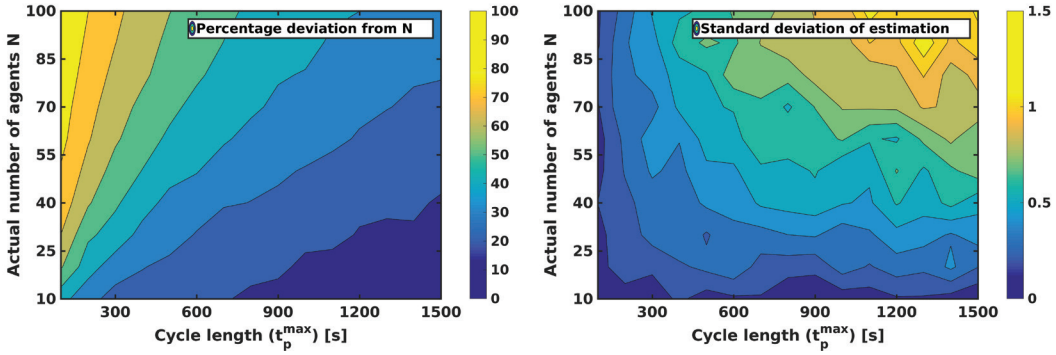
As seen in section 5, the WOSP paradigm utilizes various characteristics of the incoming pings to infer



(a) Time: $0\,t_p^{max}$ (b) Time: $1.35\,t_p^{max}$ (c) Time: $3.35\,t_p^{max}$

(d) Time: $4.35\,t_p^{max}$ (e) Time: $5.45\,t_p^{max}$ (f) Time: $5.45\,t_p^{max}$

Figure 20. Collective transport by sequential execution of gas expansion and aggregation at the object. In (a), the magenta and the grey colored patches represent the object and the goal, respectively. In (b), the agents expand in all directions to find the object and the goal. Once both the object and the goal is found, the agents that found the object (shown in blue color) exert a force on the object to move it towards the incoming pings. Since the incoming pings originate from the agents that found the goal, the object is moved towards the goal as shown in figures (c)–(e). The trajectories of the object and the agents which move the object are shown in the red and the blue lines in (f).



(a) (b)

Figure 21. (a)Percentage deviation of the estimated number of agents in the swarm versus cycle length and the actual number of agents in the swarm. Agents consistently underestimate the number of members of the swarm. The deviation decreases for decreasing N and increasing t_p^{max}.(b) The standard deviation of the estimates of population are shown with increasing N and increasing t_p^{max}. For each data point, 10 simulation runs were conducted. Parameters used: $N \in \{10, 20, .., 100\}$, $R_s = 2.33$ r, $t_{ref} = 5$ s, $t_p^{max} \in \{100, 200, .., 1500s\}$.

global level properties of the swarm or to coordinate with each other to accomplish a common goal. For example, the estimation of swarm members is done by counting the number of pings in a single cycle length. The three main characteristics of incoming pings which contribute to the working of the primitives are: (1) the direction from which pings come in, (2) the number of pings received in one cycle length and (3) the timing of the incoming pings. Based on these characteristics, an analysis can be conducted and the dependence of the WOSP primitives on various parameters can be established. Since the primitives 'estimation of swarm members', 'leader election' and 'aggregation' are dependent upon counting the number of pings, timing

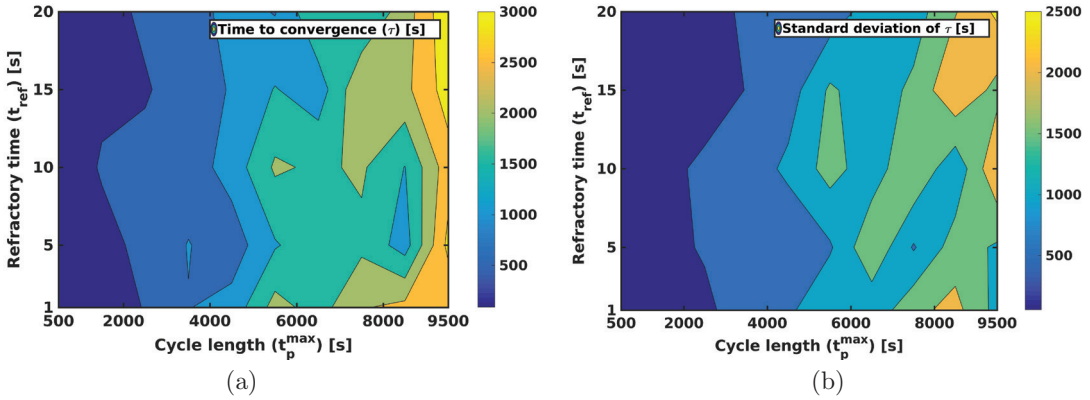


Figure 22. Time to convergence (selection of a single leader) and the standard deviation of the time to convergence is visualized in (a) and (b), respectively showing the effect of the refractory time t_{ref} and the cycle length t_p^{max}. As expected, the time to convergence (τ) rises with the increase in cycle length. t_{ref} does not seem to have a noticeable effect on τ. For each data point, 10 simulation runs were conducted. Parameters used: $N = 100, R_s = 2.33$ r, $t_{ref} \in \{1, 5, 10, 15, 20s\}$, $t_p^{max} \in \{500, 1500, ..9500s\}$.

of the incoming ping and the direction of the incoming ping, respectively, we will use these primitives as exemplary primitives to demonstrate their dependence on their parameters. From such an analysis, inferences can be drawn on other primitives which employ similar characteristics of incoming ping waves.

7.1.1. Estimation of swarm members

The dependence of the estimate of swarm members on the t_p^{max} is shown in the color map in figure 21(a), where the percentage deviation N_{err} from the actual number of swarm members N depending on the maximum possible cycle length t_p^{max} is shown. For every data point the simulation was run for $25t_p^{max}$ which was sufficiently long for the estimate to converge. The experiment was repeated 10 times for empirical validity and the standard deviation of the population estimates are shown in figure 21(b). In figure 21(a), it can be seen that for a fixed cycle length t_p^{max}, the N_{err} or the error of the estimate increases with the increase in the number of swarm members. It can also be observed that N_{err} decreases with increasing t_p^{max} for a fixed number of swarm members. Therefore, knowing the order of magnitude of N of a swarm, the t_p^{max} can be chosen to be sufficiently large such that the deviation is sufficiently small. For instance, considering a swarm of a maximum of 30 agents of the presented kind, a cycle length of $t_p^{max} > 1500$ would ensure a deviation as low as 10%. It can be therefore said in general that for primitives that are dependent on the count of incoming pings, the performance of the primitive will depend heavily on the choice of t_p^{max}.

7.1.2. Time taken to elect a leader

An analysis on time taken for the convergence (τ) of the primitive 'leader election' is performed with varying refractory time t_{ref} and cycle length t_p^{max}. For each data point, 10 simulation runs were conducted. Figure 22(a) shows that for a given number of agents, for increasing cycle length the time taken to select a leader increases. The standard deviation of the time taken for the convergence (τ) is shown in figure 22(b). In figure 22(a), it can be seen that for decreasing cycle lengths, the time taken to select a leader decreases. Therefore, the selection of cycle length t_p^{max} affects the time taken to select a single leader. Since for increasing refractory time, the time taken to select a leader remains relatively constant, when selected to be within safe limits (as defined in section 7.2), the refractory time t_{ref} does not seem to have a significant effect on the time taken to elect a leader. The standard deviation of τ, as shown in figure 22(b) is consistently within 20% of t_p^{max} which is caused by the agent timers being de-synchronized by random initialization within the range $[0, t_p^{max})$. Since the leader election primitive is dependent on the timing of incoming pings, similar conclusions can be drawn about primitives such as synchronization where timing of the incoming ping is crucial.

7.1.3. Aggregation success

The aggregation primitive has been chosen for analysis as it includes locomotion of agents and since it is dependent on the direction of incoming pings. So far we have assumed that the agents have infinite angular resolution ($\Delta\omega$), that is, they can precisely detect the direction of the incoming pings. In practise, as shown in section 8, the agents have a lower angular resolution. Therefore, we will evaluate how this assumption affects the performance of the swarm. Figure 23 shows the percentage success of the aggregation primitive being evaluated with varying lengths of steps d from 0.13 to 0.93 of the agents' sensor range and also the angular resolution $\Delta\omega$ of incoming ping direction. Since for $\Delta\omega > 120°$, the percentage convergence drops rapidly, it can be inferred that at least an angular resolution of $\Delta\omega = 120°$ is necessary for consistent and successful operation of the aggregation primitive. Smaller step sizes allow for minimizing the effects of an inaccurate movement direction and therefore enable the primitive to be successful in aggregation more often. The dependency of aggregation on $\Delta\omega$ that is found

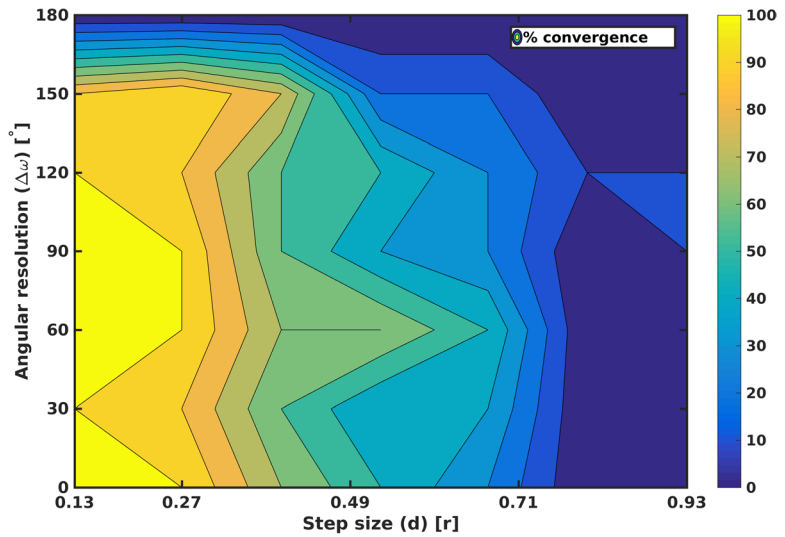


Figure 23. The percentage success of the aggregation primitive is evaluated with varying lengths of steps d taken by agents and also the angular resolution $\Delta\omega$ of incoming ping direction. It can be observed that when the length of steps taken by agents is below a third of sensor range, until about 90° angular resolution, the success of the aggregation primitive is not adversely affected. As the angular resolution decreases below 90°, the success of the primitive falls rapidly even for small step sizes. Generally, as the length of step d increases for any angular resolution, the success of the aggregation primitive suffers. Parameters used: $N = 100, R_s = 2.33$ r, $\Delta\omega \in \{30, 60, .., 180°\}, d \in \{\frac{0.4}{3}, \frac{0.8}{3}.., \frac{2.8}{3}r\}$.

Table 1. The parameters which were used for the empirical analysis of the presented primitives are shown here. Apart from the parameters previously presented, τ refers to the average time taken for each of the primitives to converge and S is the convergence rate. A set of parameters can be reliably found for the convergence of each primitive. For primitive specific parameters such as d for those primitives including locomotion, default values were used as specified in section 4.1.

Primitive	N	$tref$ (s)	R_s (r)	t_p^{max} (s)	τ (t_p^{max})	S (%)
Synchronization	100	10	2.33	1000	0.03	100
Leader election	100	10	2.33	1000	0.29	100
Localize object	100	10	2.33	1000	0.11	100
Locate swarm center	100	10	2.33	1000	0.23	100
Estimate swarm members	100	10	2.33	2500	6.17	100
Locate swarm extremities	100	10	2.33	1000	1.4	100
Aggregation	100	10	2.33	1000	0.14	100
Moving collectively	100	10	2.33	1000	20.10	100
Gas expansion	100	10	0.67	1000	2.39	100

here can be applied to other primitives which depend on the direction of the incoming ping. Therefore, it can be said that primitives such as 'localizing swarm center', 'localizing an object', 'estimate extremities of the swarm' will also be similarly affected by $\Delta\omega$.

7.2. Empirical analysis and choice of parameters

In order to empirically validate the observed collective behaviors of WOSP, each primitive was simulated 10 times with its respective convergence condition which is listed under each primitive. The results of this empirical simulation test can be found in table 1. It can be observed that primitives reliably converge for the given set of parameters. This shows that a set of parameters can be found for the convergence of various primitives of WOSP. For primitive specific parameters such as d for those primitives including locomotion, default values were used as specified in section 4.1.

Additional to the understanding gained from the analysis above, the following guidelines can be formulated based on the general understanding of the working of the WOSP paradigm. Using the time for an agent to be activated by an incoming signal s as basic temporal unit and the perception range r as basic spatial unit, the speed with which a signal propagates is approximately $v_{ping} \approx r/s$. Estimating an upper limit for the spatial extension of a swarm R_s^{max} (considering its environment and its expected maximum number of N_{max} agents) equation (3) represents the 'end-to-end signaling time'(t_{ee}), that is, the maximum time for a signal to propagate from one end of the swarm to the other. Having defined those basic swarm level quantities, the parameters cycle length t_p^{max}, refractory time t_{ref} and step size of a moving agent d can be set accordingly.

$$t_{ee} \approx \frac{R_s^{max}}{v_{ping}}. \quad (3)$$

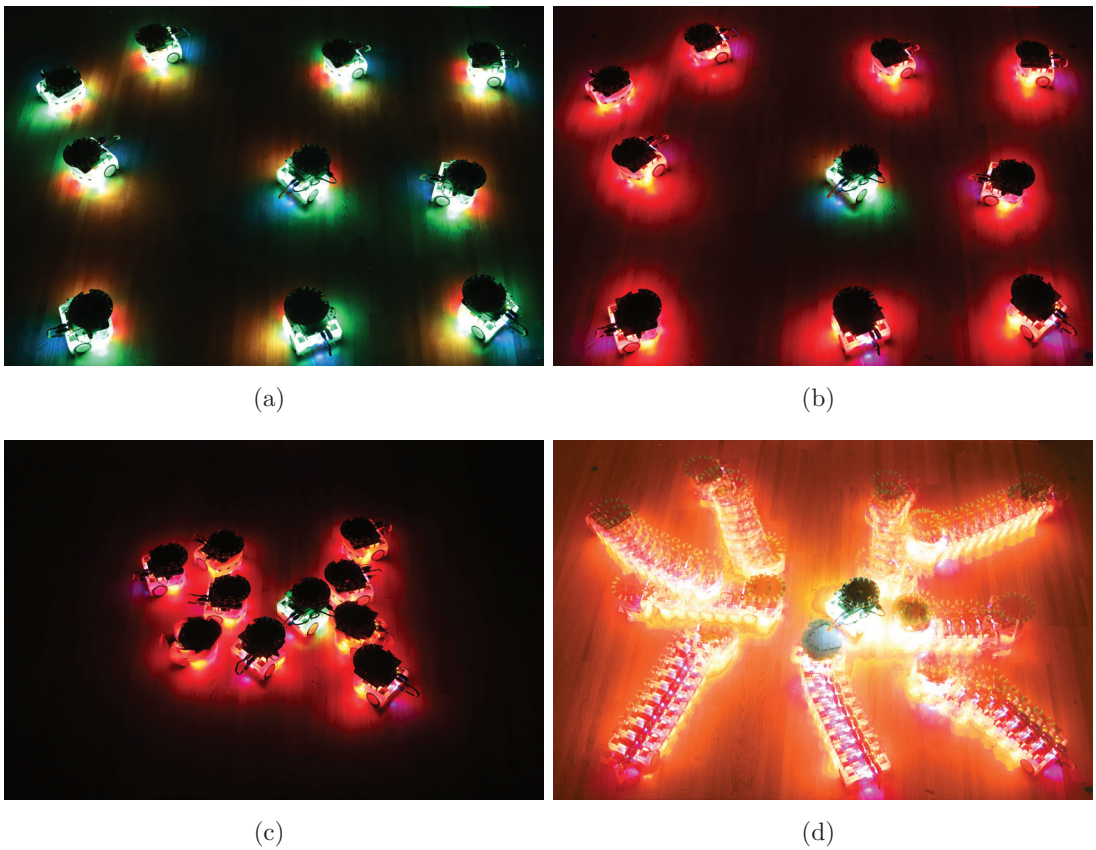


Figure 24. Photographs of various stages of WOSP validation using robotic experiments. An exemplary experiment is recorded in the following photographs. Figure (a) shows the start of an experiment. In (b), after the leader election primitive, one robot (shown in green) is elected as the leader. The non-leader robots have their red LEDs turned on. In (c), the final state of the experiment is shown where all the non-leader (red) robots have aggregated around the leader robot (green). In (d), a trace of the trajectories of the non-leader robots moving towards the leader while executing the aggregation primitive is shown using a long exposure photograph.

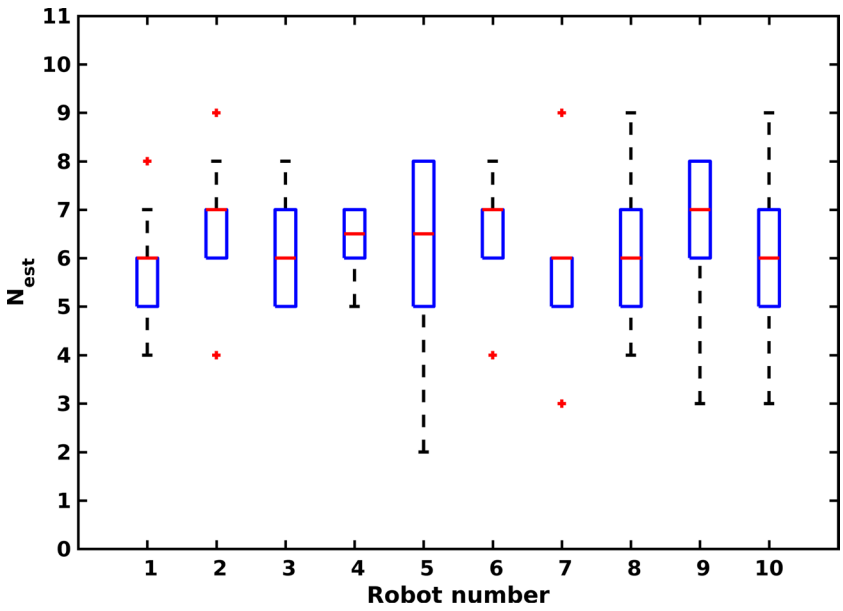


Figure 25. Plot showing estimate of population of each of the 10 robots in the swarm across 10 experiments. *X* axis shows the identity of robots $\in \{0, 1, 2 \ldots 10\}$. Parameters used: $N = 10$, $t_{ref} = 10$ s, $t_p^{max} = 300$ s.

- Cycle length: For a primitive dependent heavily on the number of incoming pings, the cycle length t_p^{max} should be large enough to allow for every agent initiate a ping wave which likely reaches all other agents without signal collisions: $t_p^{max} = N_{max} \cdot t_{ee}$. Depending on how well a swarm compensates signal collisions, the value can be adjusted accordingly.

Table 2. Table showing the parameters used in the robotic experiments and also the empirical results of the ten repetitions conducted.

Primitive	N	$t_{ref}(s)$	t_p^{max} (s)	d (r)	Repetitions	S (%)
Estimate swarm members	10	10	300	0.1	10	100
Leader election	10	10	300	0.1	10	100
Aggregation	10	10	300	0.1	10	100

- Refractory time: Generally, refractory time t_{ref} should be just large enough to make circular waves impossible by allowing enough time for a single ping wave to go through the entire swarm. In addition to t_p^{max} being large enough, an appropriate refractory time prevents splitting of groups for primitives such as leader election, aggregation etc. Therefore, we take the refractory time to be twice the time taken for a wave to travel through the swarm: $t_{ref} = t_{ee} \cdot 2$.
- Step size: Size of step d an agent will take when activated in primitives including locomotion (directly relating to speed of agents) should be chosen such that connectivity is preserved. From figure 23, it is seen that a for reliable operation, the step size should be smaller than half the sensor range, that is $d < 0.5r$.

8. Experimental validation

In order to validate the WOSP primitives which were presented so far in simulation, a swarm of robots[3] were used to execute a simple sequential combination of primitives: leader election, estimating number of swarm members and aggregation. The start of the experiment is shown in the photograph in figure 24(a). Initially, all robots were randomly distributed in the arena and each of them considered themselves candidate leaders represented by their green LEDs in 'ON' state. During the leader election primitive, within a single cycle length, all but one robot, are eliminated from leadership. All robots which were eliminated as leaders turned on their red LEDs as shown in figure 24(b). In the next two cycles, the robots estimated the number of swarm members in the swarm. The estimate of population of the individual robots over ten runs are shown in figure 25. It can be noted that the mean estimate of population for each robot (between 5 and 7) is below the actual value as expected. The deviation from the actual value of the number of swarm members is close to what was achieved in simulation as shown in figure 11. As discussed under the respective primitive, the value of t_p^{max} can be adjusted for a better estimated of the total number of members in the swarm. After estimating the number of swarm members, all the robots aggregate at the leader as shown in figure 24(c). Additional to the WOSP based aggregation found in the above sections, a simple obstacle avoidance behavior was also integrated into the aggregation primitive. The experiment was stopped when the robotic platforms were only executing obstacle avoidance behavior repeatedly. The trajectory of the aggregating robots executing the aggregation primitive is shown using a long exposure photograph in figure 24(d). In order to empirically validate the experiment results, the sequential execution of primitives was repeated ten times. All the parameters used for the experiment are shown in table 2. The parameters, t_{ref} and t_p^{max} were selected according to the selection guidelines provided in section 7.

[3] The details of the hardware used can be found in appendix.

9. Discussion

In section 5, a set of primitives is presented, which can be utilized and combined as basic building blocks for a meta control scheme for a swarm, covering the categories 'internal organization', 'swarm awareness' and 'locomotion'. Two exemplary realizations of combination of previously presented primitives for complex collective behaviors are presented in section 6. In this chapter, it is demonstrated that WOSP unifies common collective behaviors and thus enables swarms consisting of agents with limited abilities to collectively perform a large variety of complex behaviors. Due to its simple and flexible fundamental concept of 'scroll wave' based communication, this paradigm is applicable to a large spectrum of different types of swarms and environments while requiring minimal communication abilities.

9.1. General features

The primitives of WOSP inherit a set of features due to the nature of the underlying communication paradigm. The features are discussed in the following sections.

9.1.1. Unification

Evidently, many primitives such as leader election (Karpov and Karpova 2015, Ben-Shahar *et al* 2014), synchronization (Perez-Diaz *et al* 2018), aggregation (Trianni *et al* 2003, Bahgeçi and Şahin 2005, Schmickl *et al* 2008), estimating the number of members in a swarm (Melhuish *et al* 1999, Brambilla *et al* 2009) are collective behaviors that have been implemented using various mechanisms. As shown in section 5, WOSP unifies collective behaviors under a single minimialistic paradigm. Since the paradigm uses the same communication framework for all the primitives, the sequential combination of primitives can be easily accomplished as shown in section 6. Apart from the conceptual simplicity of unifying behaviors, one benefit of such unification is that the same incoming

signals can be used for inferring global properties of the swarm. For example, the number of agents in the swarm can be estimated at the same time as estimating the area of ping origin.

9.1.2. Minimalism

Minimalism is another benefit of WOSP that has been emphasized throughout the chapter. All primitives presented in section 5 use single-bit communication. Although in the real world there is seldom such a stringent limitation of using such a narrow bandwidth to communicate between robots, the extreme assumption is made in order to demonstrate that much behavioral diversity is possible with a low bandwidth. In a real implementation of WOSP, a larger bandwidth will usually be available to the programmer. With an increased bandwidth, more primitives can be added and more easily combined with each other. A larger communication payload will enable an easier parallel combination of behaviors. For example, if there are two bits of communication payload, one can be used to communicate a detected obstacle and the other for the occasional pings from agents, the swarm can go around an obstacle and still stay together by moving towards the incoming ping from agents while moving at a certain angle to the incoming bit indicating the obstacle. In essence, more complex combinations can be done by increasing the bandwidth of communication.

9.1.3. Resilience

Resilience to ping loss is a benefit WOSP inherits from using scroll waves. An analysis has been conducted on the resilience of scroll wave based communication (Varughese *et al* 2017) where its robustness against signal loss was examined. It was shown that due to redundancy in signal pathways, a system using slime mold based communication, as utilized in WOSP, can compensate up to 70% individual probability of signal loss without significant decrease in performance. The ability of this basic behavior to cope with high amounts of signal loss endows WOSP with resilient functioning when pings fail to be sent or received. A detailed analysis of this aspect can be found in Varughese *et al* (2017).

9.1.4. Scalability

As opposed to approaches such as the one presented by Le Goc *et al* (2016), decentralized control in WOSP allows scalability limited primarily by the communication abilities relative to operational time scales. For the class of swarms presented in this chapter the main constraint to scalability is constituted by the condition that maximum internal cycle length t_p^{max} must be significantly larger than the time for a ping wave to propagate from one end of the swarm to the other. It ensures that ping waves likely propagate through the entire system without colliding with other waves, thus enabling swarm-wide communication. For example, as shown in figure 21, in the primitive 'estimating the number of swarm members', the performance of the estimate is affected by the choice of t_p^{max}. Now, considering a case where t_{ee} is high due to the employment of a slow communication mechanism for agent-to-agent communication, a high t_{ee} will necessitate a high t_p^{max}. In such a case, there needs to be a trade off between the number of agents and operational time scales. However, more sophisticated techniques can be employed in addition to the simple ping counting mechanism used in section 5 for estimating the number of swarm members. Brambilla *et al* (2009) suggests a variation of Melhuish *et al* (1999) to more efficiently estimate the number of members in a swarm by changing the cycle length during the run time. Such mechanisms can be used to increase the efficiency of the primitives and improve their scalability.

9.2. Design considerations

Directives for parameter selection for primitives in general have already been discussed in section 7 and in section 4. The selection of parameters is crucial for successfully using the WOSP paradigm. However, there is a wide range of parameters for which the primitives function and a set of parameters can be reliably found for each primitive. The optimality of the selected parameter set needs to be evaluated based on the operating time scales and available hardware by the swarm programmer. For example, if the cycle length of the swarm is large enough, there will be less collisions of pings and therefore, there can be a more reliable count of the number of members in the swarm. If the time scales of operation demand a faster convergence for a given number of agents, then the programmer must resort to an alternate method for counting the number of members in the swarm. However, an analysis paying attention to physical parameters for a prospective implementation of WOSP in a robotic swarm is beyond the scope of this paper.

One of the prerequisites for a swarm to be able to implement WOSP is directional communication similar to most animals in nature which exhibit swarm behavior. In this chapter, most simulations conducted follow the assumption that agents have the ability to precisely detect the direction of incoming pings. In practice, this requirement can be substantially loosened for the agents to have a lower angular resolution without significant loss of functionality as shown in figure 23. In other words, a rough perception of the direction of the incoming ping is sufficient for the basic functionality of WOSP primitives that depend upon incoming ping direction. More specifically, primitives involving motion or directionality such as aggregation, gas expansion, collective motion, localizing an object will suffer in preciseness with a lower resolution of angular perception. For primitives that requires the agent to move in a particular direction, low resolution can be compensated with slower movement speeds.

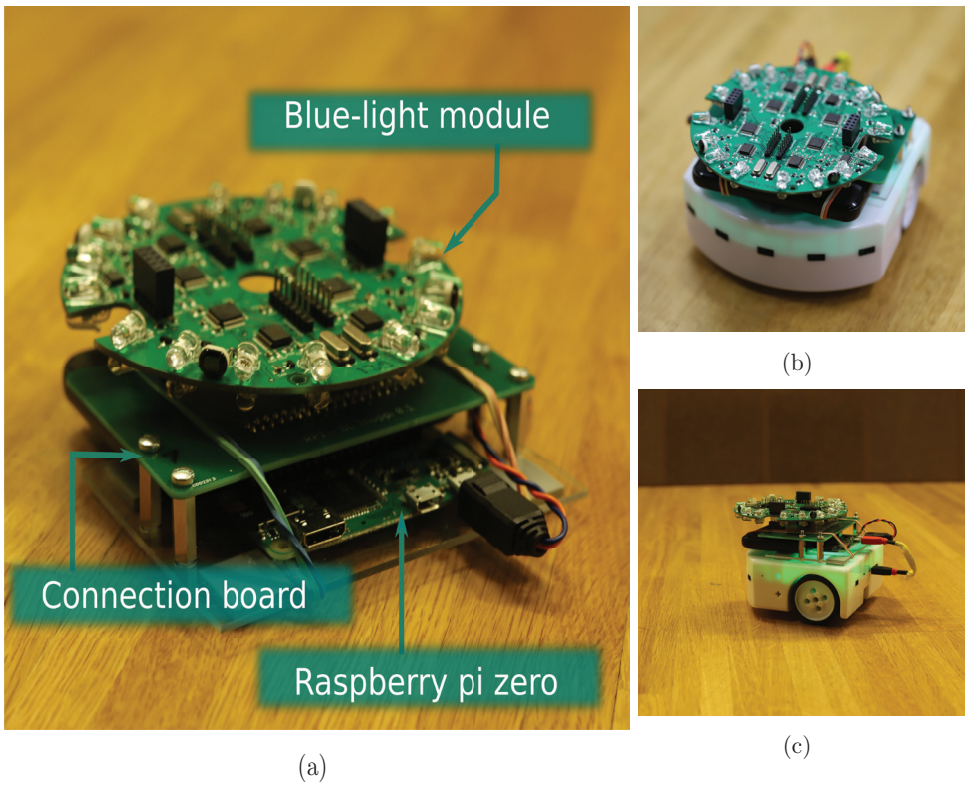


Figure A1. Photographs of electronic module integrated with thymio-II platform for directional communication. (a) Photograph shows the blue-light module which is integrated with raspberry pi zero to form the directional communication module. Figures (b) and (c) show the thymio-II robotic platform integrated with the electronics module.

As in the case of many swarm algorithms, connectivity is a prerequisite for a swarm to be able to implement WOSP. Since the focus of the paper is to present the numerous collective behaviors using minimalistic communcation, we did not discuss re-connection strategies in case of the loss of connectivity. Re-connection measures maybe needed during the actual implementation of the primitives on real robotic systems as done by 'cohesive behavior' in Bjerknes *et al* (2007). For example, in the gas expansion primitive, the agents move away from an incoming ping until it does not get any pings and then move back to reconnect with the swarm. Although such a strategy on real robots risks the robots not being able to reconnect with the swarm, it is necessary while using a single bit communication. As mentioned before, the single-bit communication is not often a requirement on real robotic platforms. On the robotic platforms used to demonstrate WOSP in section 8, the strength of the incoming signal can be detected. As an alternative to complete disconnection during gas expansion, a swarm could use the strength of the incoming signal to move away from the incoming ping while ensuring connectivity.

As mentioned in the introduction, the development of WOSP is closely associated with project subCULTron (subCULTron 2015). Within the framework of this project, individual primitives of WOSP are already being used for swarm control. Robotic systems such as subCULTron, which employ a large number of individual agents in a noisy environment aiming for autonomous operation, can benefit from WOSP. A practical application of the subCULTron swarm using WOSP is shown in Varughese *et al* (2019). In a real world implementation of WOSP, simplifying assumptions employed in this chapter will need to be addressed. For example, the speed of the communication mechanism will determine the speed with which a message travels through the swarm. The physical speed of the agents will need tuning with respect to communication speed in order to keep the swarm together. A number of other practical considerations will need to be addressed depending on the specific physical properties of the swarm. In the experiment conducted in section 8, this was not a major concern since it was not physically possible for the robots to be faster than the communication speed of the blue-light modules. In general, WOSP unifies several useful swarm behaviors under one paradigm. However, the simplifications made in this chapter to demonstrate the capabilities of WOSP need to be addressed by the programmer for each individual swarm.

Conclusively, a general unifying paradigm for common swarm behaviors which can be used to control a swarm of agents is presented in this chapter. As demonstrated through the individual primitives in this chapter, the 1-bit bioinspired communication paradigm can be used in swarm robotics to produce

a large variety of behaviors for simple robotic systems with limited capabilities and minimalistic communication. The basic primitives can also be combined to form complex behaviors using the same underlying minimalisitic paradigm.

Acknowledgments

This work was supported by EU-H2020 Project subCULTron, funded by the European Union's Horizon 2020 research and innovation programme under Grant agreement No. 640967. Furthermore this work was supported by the COLIBRI initiative at the University of Graz.

Appendix. Experimental setup

The robots were made by modifying the thymio-II robotic platform (Riedo *et al* 2013) to be able to run the WOSP code and directionally transmit and receive signals. For this purpose, the thymio-II robots were integrated with a raspberry pi (Pi 2017) and a blue-light communication module developed in project subCULTron (Thenius *et al* 2016). The raspberry pi was used to run the WOSP code and also to coordinate communications between the code, the blue-light board and the thymio-II robotic platform. The blue-light board consists of four blue light modules (a resulting angular resolution of $\Delta\omega = 90°$) which are able to send and receive 16-byte messages in their respective direction. The range of the blue-light modules used here is about one meter (Thenius *et al* 2016). The blue-light module was used to communicate using modulated blue light. Although the blue-light boards send 16-byte messages by default, the WOSP program developed here checked whether a message was received from the four different directions without considering the content of the received message in order to emulate single-bit signaling. The electronics module containing the connection board, blue-light module and the raspberry pi zero is shown in figure A1(a). Figures A1(b) and (c) show the electronics module integrated on the thymio-II platform to enable the entire robot to move and communicate directionally.

ORCID iDs

Joshua Cherian Varughese https://orcid.org/0000-0002-3250-0742

References

Abelson H, Allen D, Coore D, Hanson C, Homsy G, Knight T F Jr, Nagpal R, Rauch E, Sussman G J and Weiss R 2000 Amorphous computing *Commun. ACM* **43** 74–82

Alcantara F and Monk M 1974 Signal propagation during aggregation in the slime mould *Dictyostelium discoideum Microbiology* **85** 321–34

Bahgeçi E and Şahin E 2005 Evolving aggregation behaviors for swarm robotic systems: a systematic case study *Swarm Intelligence Symp. Proc. 2005 IEEE* (IEEE) pp 333–40

Ben-Shahar O, Dolev S, Dolgin A and Segal M 2014 Direction election in flocking swarms *Ad Hoc Netw.* **12** 250–8

Bjerknes J D, Winfield A and Melhuish C 2007 An analysis of emergent taxis in a wireless connected swarm of mobile robots *IEEE Swarm Intelligence Symp.* (IEEE) pp 45–52

Brambilla M, Pinciroli C, Birattari M and Dorigo M 2009 A reliable distributed algorithm for group size estimation with minimal communication requirements *Int. Conf. on Advanced Robotics* pp 1–6

Brooks R 1986 A robust layered control system for a mobile robot *IEEE J. Robot. Autom.* **2** 14–23

Buck J and Buck E 1966 Biology of synchronous flashing of fireflies *Nature* **211** 562–4

Camazine S, Deneubourg J L, Franks N R, Sneyd J, Theraulaz G and Bonabeau E 2001 *Synchronized Flashing Among Fireflies* (Princeton, NJ: Princeton University Press) pp 143–66

Čejková J 2015 Aggregation of slime mold *Dictyostelium discoideum* Youtube video (https://tinyurl.com/yyt7lxnk)

Christensen A L, O'Grady R and Dorigo M 2009 From fireflies to fault-tolerant swarms of robots *IEEE Trans. Evolutionary Comput.* **13** 754–66

Coore D 2004 Towards a universal language for amorphous computing *Int. Conf. on Complex Systems*

Eberhart R C, Shi Y and Kennedy J 2001 *Swarm Intelligence* (Amsterdam: Elsevier)

Fink J, Hsieh M A and Kumar V 2008 Multi-robot manipulation via caging in environments with obstacles *IEEE Int. Conf. on Robotics and Automation* (IEEE) pp 1471–6

Groß R, Tuci E, Dorigo M, Bonani M and Mondada F 2006 Object transport by modular robots that self-assemble *Proc. IEEE Int. Conf. on Robotics and Automation* (IEEE) pp 2558–64

Hayes A T, Martinoli A and Goodman R M 2003 Swarm robotic odor localization: off-line optimization and validation with real robots *Robotica* **21** 427–41

Karpov V and Karpova I 2015 Leader election algorithms for static swarms *Biologically Inspired Cogn. Archit.* **12** 54–64

Kennedy J and Eberhart R C 1995 Particle swarm optimization *IEEE Int. Conf.on Neural Networks* (IEEE Press)

Kube C R and Bonabeau E 2000 Cooperative transport by ants and robots *Robot. Auton. Syst.* **30** 85–101

Labella T H, Dorigo M and Deneubourg J L 2006 Division of labor in a group of robots inspired by ants' foraging behavior *ACM Trans. Auton. Adapt. Syst.* **1** 4–25

Le Goc M, Kim L H, Parsaei A, Fekete J D, Dragicevic P and Follmer S 2016 Zooids: building blocks for swarm user interfaces *Proc. of the 29th Annual Symp. on User Interface Software and Technology* (ACM) pp 97–109

Maes P 1989 How to do the right thing *Connect. Sci.* **1** 291–323

Matarić M J 1997 Reinforcement learning in the multi-robot domain *Robot Colonies* (Berlin: Springer) pp 73–83

Matarić M J and Michaud F 2008 Behaviour-based systems *Springer Handbook of Robotics* (Berlin: Springer) pp 891–909

Melhuish C, Holland O and Hoddell S 1999 Convoying: using chorusing to form travelling groups of minimal agents *Robot. Auton. Syst.* **28** 207–16

Middleton E J and Latty T 2016 Resilience in social insect infrastructure systems *J. R. Soc. Interface* **13** 20151022

Nagpal R 2002 Programmable self-assembly using biologically-inspired multiagent control *Proc. of the First International Joint Conf. on Autonomous Agents and Multiagent Systems: Part 1* (ACM) pp 418–25

Nagpal R, Kondacs A and Chang C 2003 Programming methodology for biologically-inspired self-assembling systems *AAAI Spring Symp. on Computational Synthesis* pp 173–80

O'Keeffe K P, Hong H and Strogatz S H 2017 Oscillators that sync and swarm *Nat. Commun.* **8** 1504

Perez-Diaz F, Trenkwalder S M, Zillmer R and Groß R 2018 Emergence and inhibition of synchronization in robot swarms *Distributed Autonomous Robotic Systems* (Berlin: Springer) pp 475–86

Pi R 2017 Raspberry pi hardware (www.raspberrypi.org/documentation/hardware/raspberrypi/README.md)
Pinciroli C, Lee-Brown A and Beltrame G 2016 Buzz: an extensible programming language for heterogeneous swarm robotics *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* pp 3794–800
Pulliam R 1973 On the advantages of flocking *J. Theor. Biol.* **38** 419–22
Reid C R and Latty T 2016 Collective behaviour and swarm intelligence in slime moulds *FEMS Microbiol. Rev.* **40** 798–806
Reynolds C W 1999 Steering behaviors for autonomous characters *Game Developers Conf.* vol 1999 pp 763–82
Riedo F, Chevalier M, Magnenat S and Mondada F 2013 Thymio II, a robot that grows wiser with children *IEEE Workshop on Advanced Robotics and its Social Impacts* (IEEE) pp 187–93
Rubenstein M, Cornejo A and Nagpal R 2014 Programmable self-assembly in a thousand-robot swarm *Science* **345** 795–9
Schmickl T, Thenius R, Möslinger C, Radspieler G, Kernbach S and Crailsheim K 2008 Get in touch: cooperative decision making based on robot-to-robot collisions *Auton. Agents Multi-Agent Syst.* **18** 133–55
Ševčíková H, Jitka Č, Lenka K, Michal P, František Š and Miloš M 2010 A new traveling wave phenomenon of *Dictyostelium* in the presence of cAMP *Physica* D **239** 879–88
Siegert F and Weijer C J 1992 Three-dimensional scroll waves organize *Dictyostelium* slugs *Proc. Natl Acad. Sci.* **89** 6433–7
subCULTron 2015 Submarine cultures perform long-term robotic exploration of unconventional environmental niches (www.subcultron.eu/)
Sumpter D J 2005 The principles of collective animal behaviour *Phil. Trans. R. Soc.* B **361** 5–22
Thenius R *et al* 2016 SubCULTron—cultural development as a tool in underwater robotics consortium for coordination of research activities concerning the Venice lagoon system *Artificial Life and Intelligent Agents* (Berlin: Springer)
Thenius R, Varughese J C, Moser D and Schmickl T 2018 WOSPP—a wave oriented swarm programming paradigm *IFAC-PapersOnLine* **51** 379–84
Trianni V, Groß R, Labella T H, Şahin E and Dorigo M 2003 Evolving aggregation behaviors in a swarm of robots *European Conf. on Artificial Life* (Springer) pp 865–74
Turgut A E, Çelikkanat H, Gökçe F and Şahin E 2008 Self-organized flocking in mobile robot swarms *Swarm Intell.* **2** 97–120
Varughese J C, Hornischer H, Thenius R, Wotawa F and Schmickl T 2019 Collective event detection using bio-inspired minimalistic communication in a swarm of underwater robots pp 634–41
Varughese J C, Thenius R, Schmickl T and Wotawa F 2017 Quantification and analysis of the resilience of two swarm intelligent algorithms *GCAI 3rd Global Conf. on Artificial Intelligence* (*EPiC Series in Computing*) (EasyChair) vol 50 pp 148–61
Varughese J C, Thenius R, Wotawa F and Schmickl T 2016 FSTaxis algorithm: bio-inspired emergent gradient taxis *Proc. of the 15th Int. Conf. on the Synthesis and Simulation of Living Systems* (MIT Press)
Zahadat P, Hahshold S, Thenius R, Crailsheim K and Schmickl T 2015 From honeybees to robots and back: division of labour based on partitioning social inhibition *Bioinspir. Biomim.* **10** 066005