Swarm Intelligence (2025) 19:333–360
https://doi.org/10.1007/s11721-025-00251-4

Decentralised construction of a global coordinate system in a large swarm of minimalistic robots

Michal Pluhacek[1,2] **· Simon Garnier**[3] **· Andreagiovanni Reina**[4,5,6,7]

Received: 29 July 2024 / Accepted: 23 June 2025 / Published online: 18 July 2025

Abstract
Collective intelligence and autonomy of robot swarms can be improved by enabling individual robots to become aware that they are the constituent parts of a larger whole and to identify their role within the swarm. In this study, we present an algorithm to enable positional self-awareness in a swarm of minimalistic, error-prone, stationary robots which can only locally broadcast messages and estimate the distance from their neighbours. Despite being unable to measure the bearing of incoming messages, the robots running our algorithm can calculate their position within a swarm deployed in a regular formation. We show through experiments with up to 200 Kilobot robots that such positional self-awareness can be employed by the robots to create a shared coordinate system and dynamically self-assign location-dependent tasks. Our solution has fewer requirements than state-of-the-art algorithms and includes collective noise-filtering mechanisms. Therefore, it has an extended range of robotic platforms on which it can run. All robots are interchangeable, run the same code, and do not need any prior knowledge. Through our algorithm, robots reach collective synchronisation and autonomously become aware of the swarm's spatial configuration and their position within it.

Keywords Self-localisation · Swarm robotics · Kilobots · Positional awareness · Minimalistic robotics

✉ Andreagiovanni Reina
andreagiovanni.reina@gmail.com

Michal Pluhacek
michal.pluhacek@agh.edu.pl

Simon Garnier
garnier@njit.edu

[1] Center of Excellence in Artificial Intelligence, AGH University of Krakow, Krakow, Poland

[2] Faculty of Applied Informatics, Tomas Bata University in Zlín, Zlín, Czech Republic

[3] Department of Biological Sciences, New Jersey Institute of Technology, Newark, NJ, USA

[4] Centre for the Advanced Study of Collective Behaviour, Universität Konstanz, Konstanz, Germany

[5] Department of Collective Behaviour, Max Planck Institute of Animal Behavior, Konstanz, Germany

[6] IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

[7] Sheffield Robotics, University of Sheffield, Sheffield, UK

1 Introduction

Coordination in natural and artificial collective systems is, fundamentally, a spatiotemporal problem. For two or more agents to coordinate their work, each must have a sense, even if imperfect, of when and where others' actions have taken place relative to its individual frame of reference. For instance, flocking in birds or schooling in fish is only possible if each individual can quickly adjust their movement (a spatiotemporal attribute) to the movements of their immediate neighbours (another spatiotemporal attribute). Even in fixed groups of interacting agents, for instance, a network of sensors and actuators in a building, the topology (spatial) and timing (temporal) of the interactions will determine the type (e.g., synchronisation, oscillation) and the quality of the collective coordination.

In the context of swarm robotics (Hamann, 2018), a field of research studying the coordination of large groups of simple autonomous robots, the temporal aspect of coordination is relatively easy to solve. Indeed, the microprocessors that control the behaviour of the robots are, essentially, clocks. They provide each robot with an internal temporal frame of reference against which it can log the events that it perceives from its environment. While individual robot clocks may drift over time, effective decentralised methods from the wireless sensor network literature have been developed to maintain synchronisation across the group (Sivrikaya & Yener, 2004). The spatial aspect, that is, where an event happens relative to a focal individual, is, however, less straightforward. Traditionally, it is solved using either an external frame of reference (e.g., a Global Positioning System) or implementing a "mental" mapping mechanism within the robot's controller (Pritsker, 1984). The former is usually very precise, but GPS signals are not always accessible to the robotic agents (e.g., when they are blocked by obstacles) and their use runs somewhat contrary to the full-autonomy goal of swarm robotics. The latter provides increased autonomy to the robotic swarm but requires significant processing power and memory storage, which may not be available on small or microscopic robotic agents.

Here, we propose an alternative approach using a fully decentralised mechanism that can be implemented in autonomous robots with limited capabilities. In particular, our approach allows a group of robots to build a global coordinate system without requiring external reference information, preset origin, or predetermined roles for the robotic agents. It is designed to work on any robotic platform, even with minimal, undirected communication abilities and noisy distance sensors. The proposed algorithm relies on the assumption that the robot swarm is deployed in a regular formation, either a rectangular or hexagonal lattice, which are formations particularly convenient for storing, charging, and transporting large swarms of robots. To demonstrate the feasibility of our approach, we implemented it using the well-known, minimalist Kilobot platform (Rubenstein et al., 2014a). We also show that it scales up to large robotic swarms by performing validation experiments with up to 200 real and 1000 simulated Kilobot units. Our approach is also potentially applicable to the domain of wireless sensor networks (Yick et al., 2008) because our robots do not move, they only locally exchange asynchronous messages with each other, acting as static interactive autonomous sensors.

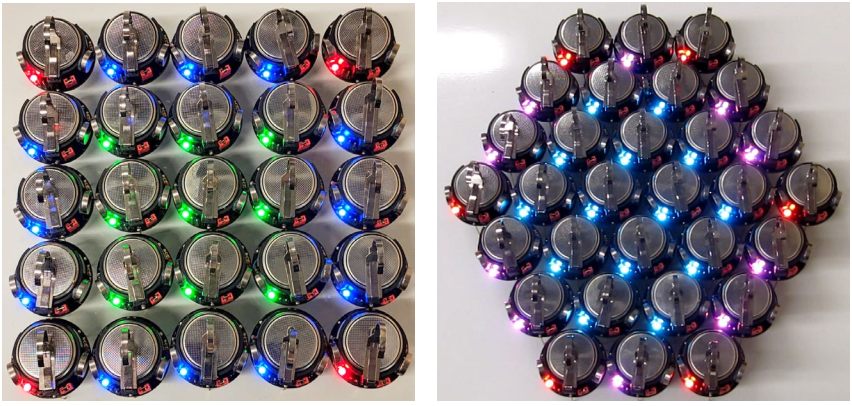
The rest of the manuscript will be organised as follows. First, we will give a general description of our approach and compare it with existing approaches in the literature, with a focus on minimalist robots (Sect. 2). In particular, we will highlight the strengths and limitations of our approach with respect to the existing ones. Then, we will provide a detailed description of the proposed algorithm and its implementation in the Kilobot platform (Sect. 3). This will be followed by a description of the results of multiple experiments

with real and simulated Kilobot swarms demonstrating the capabilities and scalability of the approach (Sect. 4), before we offer our conclusions on the possibilities that it offers and on directions for future studies (Sect. 5).

2 Previous work

Through our algorithm, robots can autonomously self-localise within a group and dynamically self-assign roles depending on their position. The algorithm works under the assumption that the robots are organised in a regular formation—either a rectangular or hexagonal lattice, as illustrated by the two examples in Fig. 1—and that they remain stationary throughout the execution. Having the robots organised in such regular formations can be particularly convenient for logistics reasons. Indeed, robots are normally stored, charged, transported, and deployed in regular formations: squared and rectangular lattices simplify the working logistics and hexagonal lattices maximise the packing density of robots with circular bodies. In addition to simplifying the transport and deployment logistics, having robots in regular formations can be a requirement for the successful execution of the collective task, for instance, the coherent motion of multi-robot aggregates (Pratissoli et al., 2019, 2023) and light pattern display (Alhafnawi et al., 2020). In these works, spatiotemporal coordination was achieved by manually providing the robots with information about their relative position within the formation. Our algorithm, by enabling the robots to self-localise, increases the system's autonomy.

There are other studies that proposed decentralised algorithms for the construction of coordinate systems and self-localisation in robot swarms. Our solution is able to work with fewer requirements and on simpler robots than state-of-the-art methods. In particular, there are a few decentralised algorithms (Beal et al., 2013; Sahin et al., 2002; Guo et al., 2011; Coppola et al., 2019; Mathews et al., 2017; Wang & Rubenstein, 2021; Batra et al., 2022; Li et al., 2018; Klingner et al., 2019; Jones & Hauert, 2025) that allow each robot



a) 5×5 rectangular lattice b) $4 \times 3 \times 4$ hexagonal lattice

Fig. 1 Two Kilobot swarms organised in two regular formations with two different topologies. In both cases, the robots by running the Routine R1 (Sect. 3.2) are able to identify their neighbourhood on the two distinct regular lattices and compute their position group, i.e., CORNER position displayed with a red light, BORDER position with a blue light in (**a**) and a magenta light in (**b**), and MIDDLE position with a green light in (**a**) and a cyan light in (**b**)

to compute its relative positioning with respect to the rest of the swarm through the use of distance and directional information about neighbours—i.e., each robot is able to know the relative location of other robots nearby—and in some cases of a global reference orientation (e.g., a compass). On the one hand, our algorithm can work on simpler robots only equipped with noisy distance sensors and transceivers for local broadcasting of small messages. Therefore, with our solution, neighbours' bearing is not needed, making its implementation on Kilobots and other minimalist robotic platforms possible. On the other hand, our solution requires the deployment of the robots in a regular formation while the robots could operate in arbitrary arrangements in the studies cited above, as long as the robots could communicate with each other.

Previous studies that implemented coordinate system construction on Kilobot swarms (hence using robots without bearing sensors) also required the robot to be deployed in a regular formation, in their case in a hexagonal lattice (Rubenstein et al., 2014b; Gauci et al., 2018). However, in their case, a subset of the robots ran a different code than the rest of the swarm and needed a precise initial placement to form the coordinate system origin and axes orientation. Another study has shown that the requirement of a regular formation can be removed by including a sizeable neighbourhood (minimum average of 15 neighbours), yet still requiring a subset of the robots to run a different code and have a precise initial placement (Nagpal et al., 2003). In our swarm, all robots run identical code and do not need any pre-configuration. Every robot can be replaced with any other and their relative position interchanged without compromising the collective behaviour. Additionally, our algorithm executes simpler mathematical operations than the previous methods by Rubenstein et al. (2014b), Gauci et al. (2018) and Nagpal et al. (2003), which can be better suited for minimal computing devices.

In summary, there is a trade-off in the requirements to run algorithms for the self-organised construction of a coordinate system. Previous works require more sensing capabilities (i.e., bearing sensors) or a subset of robots with special pre-configuration (i.e., different programs and precise placement) but can have fewer constraints on the deployment configuration of the robots. Our algorithm has reduced sensing and robot configuration requirements because it leverages the knowledge of the regular lattice configuration of the robots, which is one of the key requirements for our algorithm.

3 Self-organised construction of a coordinate system

In order to build a shared coordinate system, the robots run a sequence of three *Routines*:

R1. Neighbourhood construction
R2. Coordinate system construction
R3. Synchronised dynamic role assignment

Each routine is composed of sub-routines that we describe in detail in this section. Figure 2 gives an overview of the full process. Our three routines are designed to be run by each robot comprising the swarm. Every robot runs identical code, therefore, there is no requirement to pre-assign roles before deployment and any robot can replace any other. Despite the robots relying solely on local and error-prone communication, the execution of routines R1 and R2 allows the swarm to build a global-level coordinate system and enables every robot to self-localise within the shared coordinate system. Such capabilities can,

Routine (R1): Neighborhood identification

Sub-routine (SR 1a) Locally Unique IDs

Specific requirements:
- Communication.

Output:
- Each robot has locally unique ID.

Sub-routine (SR 1b) Neighborhood list creation

Specific requirements:
- Locally unique IDs (SR 1a).

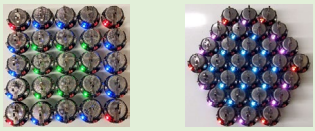
Output:
- Each robots has a list of its neighbors .

Sub-routine (SR 1c) Relative position identification

Specific requirements:
- Locally unique IDs (SR 1a) and neighborhood list (SR 1b).
- Robots physically organized in a regular lattice.

Output:
- Each robot is aware of its relative position in the lattice (e.g. corner, border or inside).

Routine R1 output:



Routine (R2): Global coordinate system

Sub-routine (SR 2a) Axes origin and direction

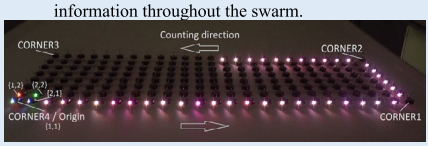
Specific requirements:
- Neighborhood identified (R1).
- Coordinate system rules specified.

Output:
- The swarm selects which robot is located at the global axes origin.
- The swarm determines the axes orientation as physical directions on the lattice.

Sub-routine (SR 2b) Lattice dimensions measurement

Specific requirements:
- Axes origin and direction (SR 2a)
- Robots physically organized in a rectangular lattice.

Output:
- Edge robots count the axes sizes and spread the information throughout the swarm.



Sub-routine (SR 2c) Coordinates assignment

Specific requirements:
- Lattice dimensions (SR2b)
- Coordinate system definition.

Output:
- Every robot is assigned a unique pair of (x and y) coordinates that correspond to the global physical position within the lattice formation.

Routine R2 output:

Routine (R3): Dynamic role assignment

Sub-routine (SR 3a) Synchronized pattern emergence

Specific requirements:
- Globally unique IDs (R2).
- Position awareness (R2).

Output:
- The swarm is able to perform any distributed computation on a physically embedded lattice.
- The swarm is kept in sync.

Routine R3 output:

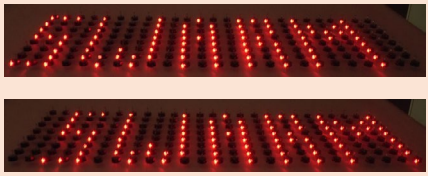


Fig. 2 Overview of the three routines to (R1) allow every robot to identify its lattice neighbourhood, (R2) construct a swarm-level coordinate system, and (R3) dynamically assign roles to robots in specific positions within the formation. All routines consist of decentralised algorithms to let the robot reach global coordination in a self-organised way. Each routine comprises one or more sub-routines (SR). The specific requirements for each subroutine also include requirements from previous subroutines

then, allow the robots to self-assign roles based on their position within the swarm and on a shared time clock, as showcased with routine R3. The routines' algorithms are presented in a generic form and are designed to work on both rectangular and hexagonal regular lattices, with exception of routine R2 which has been tailored to build a coordinate system on rectangular lattices only. In fact, the coordinate system in rectangular and hexagonal lattices is structurally different (Snyder et al., 1999), therefore, routine R2 needs to be designed specifically for each type of lattice. For the purpose of this study, we show the construction of the coordinate system (R2) in rectangular lattices only.

Before describing the three routines, we explain how the robots can maintain a time clock synchronised among all the robots, which is a fundamental requirement for the successful execution of our routines.

3.1 A synchronised swarm

In our study, every robot is an autonomous entity that runs the same code and exchanges messages with close neighbours. In several parts of the process, the transition from one subroutine to the next one is based on a shared time clock. However, without any closed-loop control, the independent clocks of a large number of devices can desynchronise over time. On the contrary, implementing a communication protocol that maintains every device in step with the same global clock at all times can require either a global orchestrator or fast communication and large bandwidth (Sivrikaya & Yener, 2004; Li & Rus, 2006). Therefore, in our system, we implement a hybrid solution similar to the method proposed by Naz et al. (2016), that combines independent clocks and closed-loop synchronisation: the robots independently measure time during limited periods and periodically exchange synchronisation messages to reset their clocks and avoid large drift.

Individual clocks In our implementation, Kilobots are equipped with a microcontroller that runs the robot's control algorithm at a rate of approximately 32 Hz. The approximate time elapsed from an event (e.g., the start of the process) can thus be measured using the number of control loops that the robot executed (which in the Kilobot's firmware are expressed as `kilo_ticks`). This approach is easy to implement and does not require communication among the robots, however, despite calibration by the producer, the Kilobots' clocks can quickly desynchronise over time. After a few minutes, the difference can become noticeable (i.e., one or two seconds). Therefore, in our implementation, robots use their internal clock to estimate the time elapsed from the beginning of the current phase of the process (e.g., a subroutine) and synchronise their internal counter at every phase change.

Message-based synchronisation When a robot reaches the given time limit according to its internal clock, it immediately moves to the next phase of the algorithm and broadcasts a synchronisation message to notify its neighbours that the new phase has begun. This message-passing strategy mirrors distributed time synchronization schemes in wireless sensor networks, which propagate timing information via local broadcasts rather than relying on a single master clock (Sivrikaya & Yener, 2004; Li & Rus, 2006). In contrast to approaches that require a global orchestrator or continuous high-rate communication to keep all devices in lockstep, our method leverages periodic local resets: any robot that receives a synchronisation signal will promptly transition to the next phase and relay the message onward. As every phase is relatively short, the clock drift accumulated between these synchronisation events remains small, consistent with prior findings that frequent re-synchronisation bounds the divergence between node clocks (van Greunen & Rabaey,

2003). In large swarms this often results in multiple robots reaching the phase deadline almost simultaneously (before receiving others' signals), so synchronisation messages are generated from multiple sources in parallel. Such a multi-source broadcast approach to time sync is known to reduce propagation delay across the network compared to a single-source scheme (van Greunen & Rabaey, 2003). This principle—allowing any node to act as a time reference—has also been observed in biologically inspired algorithms (e.g., firefly clock synchronization) and other fully-decentralized protocols (Tyrrell et al., 2006), underscoring the robustness of our design. Overall, our Kilobot implementation follows the same fundamental principles as these established distributed synchronization methods (message propagation and periodic correction) but adapts them to minimalistic robots with severely constrained hardware. Notably, recent work in multi-robot systems has shown that even simple, low-cost robots can achieve emergent temporal coordination through local message exchanges (Barciś & Bettstetter, 2020), which further situates our approach within the broader literature on distributed time synchronisation in resource-constrained systems. We validated this message-based scheme on swarms of up to 200 real and 1000 simulated Kilobots, and observed that the swarm did not desynchronise over extended periods nor accumulate any notable delay, demonstrating that classical wireless sensor network synchronization principles can be successfully applied to minimalistic robot swarms.

3.2 Routine R1—neighbourhood construction

The routine *Neighbourhood construction* (R1) aims to enable each robot to construct a list of its closest neighbours which can be identified with locally-unique IDs. The routine R1 is composed of three subroutines: SR1a Locally-unique ID assignment, SR1b Neighbour list creation, and SR1c Relative position identification.

3.2.1 Subroutine SR1a—locally-unique IDs assignment

A locally-unique ID is essential for neighbourhood construction as it allows robots to count the number of robots in their proximity and establish one-to-one communication with each neighbour. This subroutine is composed of two phases presented as pseudocode in Algorithm 1. During the first phase, each robot generates a random ID that it draws from a given range. In our case, the Kilobots use an 8-bit integer in the range [0255]. The robots broadcast their randomly selected ID. Each received ID is added to a blacklist of already used IDs. If a robot receives a message with an ID equal to its own ID, it selects a new random ID from the same range, excluding the IDs in the blacklist. After sufficient time has elapsed, in our case the time necessary to send about 20 messages (10 s), see line 1 of the Algorithm 1, the subroutine moves to the second phase. In this phase, the robots aim to remove duplicated IDs among their neighbours who may not be in direct communication with each other. However, a robot with two neighbours that use the same ID cannot distinguish between them and this causes problems in subsequent subroutines. In this second phase, each robot generates a new random number and repeatedly broadcasts messages containing four pieces of information: its ID, the just-generated random number, and the ID and the random number received from one of its neighbours. Each new message contains information about a different neighbour, iteratively selected (lines 21–22). For each received message, the robot stores the received neighbour's ID and the additional random number in a list. The use of the additional random number is necessary to distinguish between its own ID included in the neighbour's message or the ID from another robot with

the same value. The probability of random selection of both the same ID and the same random number reduces exponentially with the range size. For instance, for the Kilobots using two 1-byte numbers, the probability that two robots pick the same two numbers is smaller than 0.002%, as there are 2^{16} possible combinations. When a robot receives a message in which the third piece of information is equal to its ID and the fourth piece of information is not equal to its random number, it means that there are repeated IDs (line 23). Therefore, the robot adds its ID to the blacklist and self-assigns a new random ID. After sufficient time has elapsed, in our case the time to send about 30 messages (15 s, line 2), the subroutine SR1a terminates and the subroutine SR1b begins.

3.2.2 Subroutine SR1b—neighbour list creation

In order to create a list of its neighbours, a robot must filter incoming messages based on the senders' distance. In our implementation, the Kilobots exchange messages through infrared (IR) messages and can estimate the sender distance using the IR signal strength Rubenstein et al. (2012). In agreement with the Kilobots' capabilities, we assume that the robots can only rely on distances without directional information, that is robots cannot know the relative angle of the sender, they can only know its distance.

Subroutine SR1b is composed of three phases, and its pseudocode is shown in Algorithm 1. In the first phase, every robot continues, as it did during SR1a, to broadcast its locally-unique ID. Each robot measures the distance of the incoming messages and records the shortest distance x (erroneous values smaller than the robot body-size are discarded, e.g., 33 mm for the Kilobots). After a threshold time, the robots start the second phase. In our case, we combined the first phase of SR1b with the second phase of the subroutine SR1a (line 27 of Algorithm 1), to speed up the process. Based on the shortest distance recorded, the robot computes the maximum neighbourhood radius r which it uses to filter the incoming messages and create the neighbour list. A robot only adds to its neighbour list the ID of senders at a distance smaller than r. Computing the radius r as a function of the estimated shortest distance x allows the algorithm to adapt to different spacing and grid layouts.

Subroutine SR1b enables the robots to create their neighbour list when they are organised in the regular lattice topologies of rectangular lattices (as in Fig. 1a) and equilateral triangular lattices, also known as hexagonal lattice (as in Fig. 1b). For the case of hexagonal structures, computing r is easier as all robot neighbours are at approximately the same distance. Therefore, r can be set to $r = (1 + \epsilon)x$, where $\epsilon < 0.5$ is the proportion of tolerable error in placement/sensing. For the case of rectangular lattices we want to identify the Moore neighbourhood, therefore computing r is more difficult as it must include robots at different distances: the neighbours on the diagonal are farther than the ones at the sides. Additionally, the rectangular lattices can have different vertical and horizontal distances, that is the lattice's rows can be closer than the columns are, or vice versa. Given the robots' limitations (i.e., absence of directional information), subroutine SR1b can only work with an asymmetry between vertical and horizontal distances up to a given limit. The limit is given by the inequality $2x > \sqrt{x^2 + y^2}$, where x and y are the inter-robot distances on the two dimensions (rows and columns). Assuming $x \leq y$, the inequality states that twice the distance of one dimension should be larger than the distance on the diagonal of the rectangular grid. Thus, we have the constraint that $y < \sqrt{3}x$. If we also assume a placement/sensing error ϵ, the inequality becomes $2x(1 - \epsilon) > \sqrt{(x + \epsilon\, x)^2 + (y + \epsilon\, y)^2}$, resulting to

$$y < \frac{x\sqrt{3\epsilon^2 - 10\epsilon + 3}}{\sqrt{\epsilon^2 + 2\epsilon + 1}}. \tag{1}$$

Exceeding this limit of horizontal–vertical asymmetry—in either direction—would cause robots located two hops apart along one axis of the lattice (e.g., the x-axis) to be closer than their direct neighbours along the other (e.g., the y-axis), thereby invalidating our distance-threshold-based method for identifying neighbours.

These two limits also constrain the range of suitable values for the distance threshold r, which must lie between them, i.e., $2x > r > \sqrt{x^2 + y^2}$ (and similarly when accounting for the sensing error ϵ), as illustrated in Fig. 3. As the horizontal–vertical asymmetry increases, the valid range for r narrows, making precise tuning more critical, especially in the presence of sensing noise ϵ. The Kilobot algorithms benefit from simple computation, thus, in our implementation, the Kilobots compute $r = 1.5x + 10$, a straightforward formula that approximately lies midway between the two bounds of the inequality and consistently performed well for both triangular and rectangular lattices (see Fig. 1).

Occasionally, it might happen that due to noise in the transceiver (especially in swarms with small spacing between individual robots), the distance of the sender is consistently overestimated. In such instances, messages sent by a valid neighbour are discarded because the estimated distance is larger than r. In this case, robots may create an incomplete list of neighbours. Even if a consistent overestimation of distance by a robot is rare, the presence of these errors is relatively high when operating with large swarms because the probability of these rare events occurring increases with the swarm size. Therefore, we implemented a repairing mechanism as the third phase of the subroutine SR1b, which begins 5 s after the start of the second phase. During this repairing phase, each robot broadcasts a message with its ID (sender-ID) and all IDs in its neighbour list (neighbours-IDs). Each robot checks all incoming messages (regardless of the sender's distance) and if its ID is one of the neighbours-IDs, it adds the sender-ID to its neighbour list. This approach reduces considerably distance estimation errors which are typically asymmetric, that is only one robot in a couple of neighbours overestimates the distance.

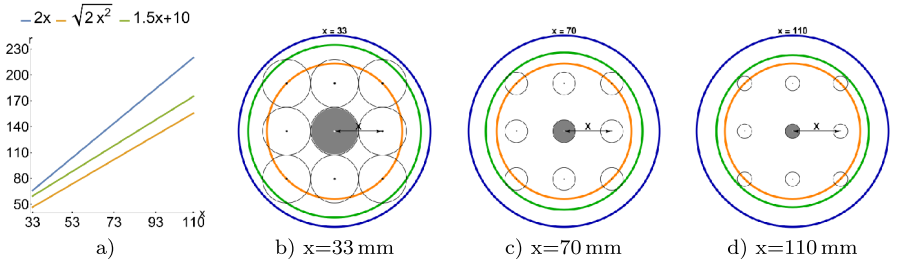


Fig. 3 In order to correctly identify the lattice neighbours (e.g., the Moore neighbourhood in a rectangular lattice) each robot filters the incoming messages using the estimated distance of the message's sender (subroutine SR1b). The filtering threshold r must be within two limits, illustrated in the figures by the yellow and blue lines, for the case of noiseless measurements and positions. The limits become more stringent when we consider placement or measurement errors, as discussed in the text. **a** In our implementation, the Kilobots use a simple equation (green line) that lays approximately in the middle of the two limit lines. The panels **b–d** show how the focal robot, in the centre, scales its threshold r, green line, when inter-robot spacing increases

Algorithm 1 Subroutines SR1a and SR1b: Locally-unique IDs assignment and neighbour list creation

Set $time\_limit_1 = 300$ ▷ Algorithm parameter (approx. 10 s)
Set $time\_limit_2 = 800$ ▷ Algorithm parameter (approx. 25 s)
Set $time\_limit_3 = 1600$ ▷ Algorithm parameter (approx. 53 s)
Set $min\_msg\_distance = 255$ ▷ Minimum distance variable
Set $i = 1$ ▷ Iterator
Set $neighbours\_count = 0$ ▷ Initialisation of counter for neighbours
Set $my\_id = Random()$ ▷ Select random ID from allowed range
Set $\overrightarrow{my\_msg} = \{my\_id\}$ ▷ Set the content my outgoing message $\overrightarrow{my\_msg}$
for each message $\overrightarrow{m_y}$ received **do**
  Set $received\_id = m_y[1]$ ▷ Store the ID of my neighbour
  Set $msg\_distance = distance\_estimate(m_y)$ ▷ Compute the distance of my neighbour
  if $kilo\_ticks \leq time\_limit_1$ **then**
    Add $received\_id$ to $\overrightarrow{black\_list}$
    if $received\_id == my\_id$ **then** ▷ 1st phase of SR1a
      Set $my\_id = random() \notin \overrightarrow{black\_list}$ ▷ Select random ID from allowed range excluding black-listed IDs
      Set $\overrightarrow{my\_msg} = \{my\_id\}$ ▷ Update the content of my outgoing message
    end if
  end if
  if $kilo\_ticks > time\_limit_1$ AND $kilo\_ticks \leq time\_limit_2$ **then** ▷ 2nd phase of SR1a
    Add $received\_id$ to $\overrightarrow{id\_list}$
    Set $\overrightarrow{my\_msg} = \{my\_id, id\_list[i]\}$
    Set $i{+}{+}$; ▷ Iterate throughout the received ids stored in the $\overrightarrow{id\_list}$
    if $my\_id \in \overrightarrow{m_y}$ **then**
      Add my_id to $\overrightarrow{black\_list}$
      Set $MY\_ID = Random() \notin \overrightarrow{black\_list}$
    end if
    if $msg\_distance < min\_msg\_distance$ AND $msg\_distance \geq robot\_body\_length$ **then**
      Set $min\_msg\_distance = msg\_distance$ ▷ Store the minimum distance value
    end if
  end if
  if $kilo\_ticks > time\_limit_2$ AND $kilo\_ticks \leq time\_limit_3$ **then** ▷ 2nd phase of SR1b
    if $received\_id \notin neighbours\_list$ **then** ▷ Avoid double counting
      if $msg\_distance < 1.5 \cdot min\_msg\_distance{+}10$ **then** ▷ Distance smaller than r
        Add $received\_id$ to $\overrightarrow{neighbours\_list}$ ▷ Store neighbour's ID in neighbours list
        Set $neighbours\_count{+}{+}$
      end if
    end if
  end if
end for

3.2.3 Subroutine SR1c—position group identification

Once the neighbour list is created, subroutine SR1c enables every robot to determine its position group in the lattice (Algorithm 2). Each robot broadcasts messages indicating its ID and the number of neighbours in its list. Once the robots have collected the information regarding the number of neighbours from every neighbour, they use it to determine their own position in the lattice. We consider three possible position groups: CORNER, BORDER, and MIDDLE, as illustrated with colour-coded positions in Fig. 1. The robot is positioned on the CORNER of the lattice if it has fewer neighbours than any of its neighbours

(line 17 of Algorithm 2). The robot is positioned in the MIDDLE if its number of neighbours is the largest of its neighbourhood (line 23). The robot is positioned on the BORDER of the lattice in all other cases (line 20, i.e., it has neighbours with a higher number of neighbours and it has not fewer neighbours than any of its neighbours). Note that for a known regular topology, this procedure can be simplified as the position in the formation can be derived directly from the number of identified neighbours (e.g., in a rectangular topology the CORNER has 3 neighbours, the BORDER has 5 neighbours, and MIDDLE has 8 neighbours). However, subroutine SR1c, and more generally routine R1, does not require the robots to know the lattice topology in advance. On the contrary, during SR1c, the robots can self-deduce the regular topology they are part of by using the neighbour counts.

Algorithm 2 Subroutine SR1c: Position group identification

```
1: Set my_msg = {my_id,neighbours_count}        ▷ Set the content of my outgoing message
2: Set max_count = 0                                                  ▷ Internal variable
3: Set min_count = 255                                                ▷ Internal variable
4: for each message m_y^→ received do
5:     Set received_id= m_y[1]
6:     if number of neighbours with unknown neighbours_count > 0 then
7:         if received_id ∈ neighbours^→ then
8:             Set received_neighbour_count= m_y[2]
9:             if received_neighbour_count < min_count then
10:                Set min_count = received_neighbour_count
11:            end if
12:            if received_neighbour_count > max_count then
13:                Set max_count = received_neighbour_count
14:            end if
15:        end if
16:    end if
17:    if neighbours_count < min_count then
18:        Set my_position=CORNER
19:    end if
20:    if neighbours_count ≥ min_count AND neighbours_count < max_count then
21:        Set my_position=BORDER
22:    end if
23:    if neighbours_count==max_count then
24:        Set my_position=MIDDLE
25:    end if
26: end for
```

3.3 Routine R2—coordinate system construction

Routine R2 enables the swarm to create a global coordinate system in which every robot self-localises by computing its 2-dimensional coordinates within the lattice. This routine requires the completion of routine R1 by which robots know the locally-unique IDs of their neighbours and know their position group in the lattice (i.e., CORNER, BORDER, or MIDDLE). This routine has been designed for rectangular lattices, however, our subroutines can be potentially extended to different protocols for building global coordinate systems (possibly in dimensions higher than two) for different lattice topologies, e.g., hexagonal (Snyder et al., 1999). For the rectangular lattice, the 2D coordinates are computed

with respect to the x and y axes which correspond to two orthogonal edges of the rectangle (that are randomly chosen in subroutine SR2a). The coordinates start from value (1,1) at the origin (a robot in a corner of the lattice) and indicate the discrete positional values for every robot in the lattice.

3.3.1 Subroutine SR2a—axes origin and direction

Through subroutine SR2a, one of the four corners is randomly selected as the axes' origin (Algorithm 3). At the beginning of this subroutine, every CORNER selects a random number, and the CORNER that selected the smallest number is selected as the axes' origin. Because the four CORNER robots are not in direct communication range, all robots cooperate in the communication by spreading the corners' random numbers throughout the lattice. In order to minimise bandwidth usage and speed up the information spreading, every robot only relays the lowest random number that it has received so far and ignores any other higher number (lines 19-21). The CORNER robots also stop broadcasting their random number once they receive a message with a number lower than their own (line 16). After sufficient time, which we estimate as the time necessary to send 25 messages (about 13 s), we assume that the message representing the lowest random number of one CORNER robot has reached all other CORNERs. The CORNER robot that has not received any lower random number, then, takes on the role of origin of the axes and sets its coordinates to (x,y)=(1,1).

In order to minimise the probability that two random numbers have the same value, the random number should be uniformly drawn from the largest range possible. For example, in our implementation, the Kilobots can exchange messages with a 9-byte payload (72 bits). Therefore, the CORNER computes its random number in the range $[0, 2^{72}]$. Using such a large range, the probability of two identical 9-byte sequences being generated in two CORNER robots is negligible.

Once the axes' origin CORNER robot is selected, it is possible to determine the axes' orientation by assigning coordinates to its neighbours. The axes' orientation is also randomly determined, this time using information already locally available (from SR1a). The CORNER robot assigns to the BORDER node with the lowest locally-unique ID the coordinate (x,y)=(2,1) and to the BORDER node with the highest ID the coordinate (x,y)=(1,2). As a result, both the origin and orientation of the coordinate system are random and self-emergent in every run. This operation terminates subroutine SR2a and enables the start of subroutine SR2b.

Algorithm 3 Sub-routine SR2a: Axes origin and direction

```
Set time_limit_4 = 400                                  ▷ Algorithm parameter (approx. 13 s)
Set coord = {0, 0}                                      ▷ Coordinates pair
Set lower_id_border = 0                                 ▷ Internal variable
Set origin = FALSE                                      ▷ Internal variable
Set start_time= kilo_ticks                              ▷ Current value of kilobot internal clock
if my_position == CORNER then
    Set origin = TRUE
    Set my_random_num = Random()
    Set my_msg = my_random_num
end if
for each message m_y received do
    if kilo_ticks < start_time + time_limit_5 then
        if my_position == CORNER then
            if m_y[1] <my_random_num then
                Set origin = FALSE
                Set my_msg = ∅
            end if
        else
            if m_y[1] < min_received_IDs[1] then
                Set my_msg = m_y
            end if
        end if
    else
        if origin==1 then
            Set coord = {1, 1}
            Set lower_id_border = ID of border node among neighbours with lower ID value.
            Set my_msg = {my_id,1,1,lower_id_border}
        end if
        if my_position == BORDER AND m_y[2] == 1 AND m_y[3] == 1 then
            if m_y[4]==my_id then
                Set coord = {2, 1}
            else
                Set coord = {1, 2}
            end if
        end if
    end if
end for
```

3.3.2 Subroutine SR2b—lattice dimension measurement

The subroutine SR2b enables the swarm to compute collectively the dimension of the two axes and gives information to the border robots on their position with respect to these axes. Knowing the axes' dimensions also allows every robot to measure the size of the swarm, a quantity that is typically hard to compute with decentralised algorithms (Saha et al., 2021; Ganesh et al., 2007).

Once the axes' origin has been selected (SR2a), the first BORDER robot on the x-axis, that is the robot with coordinates (x,y)=(2,1), starts the subroutine SR2b. The robot initialises a counter variable with the value 2 and sends this value throughout the lattice border; at each message hop, the next border robot increases the count by one and relays the message, as displayed in Fig. 4. These messages are flagged with a specific header denoting the border-count subroutine SR2b. These messages are composed of

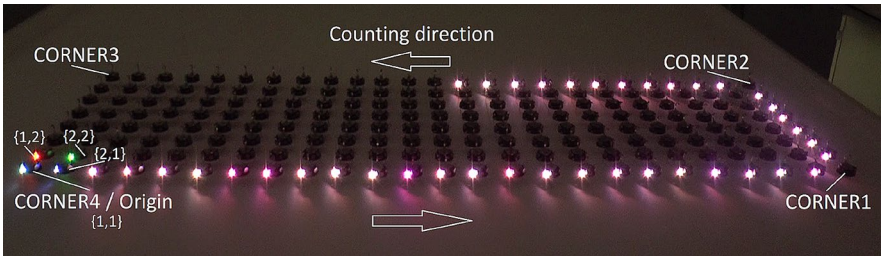


Fig. 4 Image of the border counting process (subroutine SR2b) in a swarm of 200 Kilobots. The robots on the borders of the 25×8 lattice light up when they receive the border-count message during the execution of subroutine SR2b. In this way, it is possible to see the message travelling throughout the lattice edges. In the bottom-left corner, there is the robot at the axes' origin, (x,y)=(1,1), and its neighbours display different colours to signal their coordinates as discussed in subroutine SR2a

four values (in addition to the header); one value is the border count that is increased at each step, while the other three values are initialised to zero and will be edited by the CORNER robots. The algorithmic implementation for this process is shown in Algorithm 4 and indicates that every BORDER robot that did not receive a border-count message yet accepts the message, adds one to the counter, and broadcasts the new value. Three additional rules are necessary for computing correctly the lattice edge dimension. The first additional rule regards BORDER robots that are adjacent to a CORNER robot and received their first message from another BORDER robot. These robots must use a special header in their border-count message. Messages with such special header are only read by CORNER robots so that the other BORDER robot on the diagonal (which did not receive any border-count message yet) will ignore the message and wait for the CORNER robot to send its border-count message with the updated value. The second additional rule requires the CORNER robots to append further information to the border-count message. They include information about their local count that they store in the first zero value composing the message. Therefore the border-count message maintains as separate pieces of information the local count of the three corners traversed during the multi-hop spreading. The third additional rule prevents the execution of the counting in both directions. The rule only applies to the robots with coordinates (x,y)=(1,1) and (x,y)=(1,2); these two robots ignore any border-count message with a count value lower than three. This process ends when the count has travelled around the whole border of the lattice; this happens when a valid border-count message is transmitted to the axes' origin. This process is independent of timers and can scale to any grid size larger than or equal to 3×3.

Once the origin robot receives the total border count, the total count information and the local counts of the CORNER robots are spread throughout the border in the same direction as before until the message is returned to the origin once again. At this point, all BORDER and CORNER robots know the lattice size (total border count and corners' local counts) and their position on the border with respect to the axes' origin (their local count value). The robots can use this information to compute the swarm size and will subsequently use it to compute their coordinates in subroutine SR2c.

Algorithm 4 Sub-routine SR2b: Lattice dimensions measurement

```
Set my_count = 0                                   ▷ Local variable for lattice dimension counting
if coord == {1, 1} then
    Set my_count = 1
end if
if coord == {2, 1} then
    Set my_count = 2
end if
Set my_msg = {my_id,my_count}
for each message m_y received do
    if my_position == BORDER AND my_count == 0 AND m_y[2] > 1 AND coord ≠ {1,2} then
        Set my_count = m_y[2] + 1
        Set my_msg = {my_id,my_count}
    end if
    if coord == {1,2} AND my_count == 0 AND m_y[2] > 3 then
        Set my_count = m_y[2] + 1
        Set my_msg = {my_id,my_count}
    end if
    if origin == 1 AND m_y[2] > 3 then
        Send sync signal for next phase
    end if
end for
```

3.3.3 Subroutine SR2c—coordinates assignment

Through subroutine SR2c, every robot computes its two coordinates and becomes aware of its position within the full formation. In subroutine SR2c, robots employ the information gathered in SR2b and incrementally assign coordinates, commencing from the edges of the lattice and moving inwards.

Corner and borders. The corner at the origin already has its coordinates, assigned during subroutine SR2a, (x,y)=(1,1). The other three CORNER and all the BORDER robots use the information exchanged in the border-count messages in SR2b to set their coordinates. The border-count messages contain ordered information on the local count of the three corners encountered during the multi-hop count on the border. We label as C1, C2, and C3, the local counts of the first, second, and third corners encountered during the border-count process (see Fig. 4). The CORNERs and the BORDERs then assign their coordinates through Algorithm 5, where *my_count* is the local count of the robot running the algorithm and *coord* its coordinates.

Algorithm 5 Part of subroutine SR2c: Coordinates assignment for CORNER and BORDER robots

```
if my_position == CORNER OR my_position == BORDER then
    if my_count ≤ C1 then
        Set coord = {my_count, 1}
    end if
    if my_count > C1 AND my_count ≤ C2 then
        Set coord = {C1, my_count − C1 + 1}
    end if
    if my_count > C2 AND my_count ≤ C3 then
        Set coord = {C1 + C2 − my_count, C2 − C1 + 1}
    end if
    if my_count > C3 then
        Set coord = {1, C2 + C3 − C1 − my_count + 1}
    end if
end if
```

Middle robots. The MIDDLE robots—which are not on the edge of the lattice—assign their two coordinates independently using information locally broadcast by their neighbours. Once any robot self-assigns its coordinates, it broadcasts its values to its neighbours. Each robot locally stores the coordinates of its neighbours and once it receives the coordinates with three consecutive values on one axis, it self-assigns the middle value on that axis. For instance, a robot i that receives the coordinates from neighbours j, k, and w with values $(x_j, y_j) = (3, 7)$, $(x_k, y_k) = (4, 7)$, and $(x_w, y_w) = (5, 7)$, will set its coordinate $x = 4$. Similarly, when a robot i receives the coordinates, $(x_j, y_j) = (3, 7)$, $(x_k, y_k) = (3, 8)$, and $(x_w, y_w) = (5, 9)$, will set its coordinate $y = 8$. This process allows every robot to self-assign its coordinates and to become aware of both its position within the lattice and the relative position of all its neighbours. This information can be employed for various subsequent tasks which exploit spatial awareness of the robots (e.g., dynamic role assignment as a function of the robots' position (Pratissoli et al., 2019, 2023), or exploit the swarm-level agreement obtained with routine R2 (e.g., use the coordinates to assign globally-unique IDs).

3.4 Routine R3—synchronised dynamic role assignment

Routine R3 consists in assigning a different role, or task, to each robot depending on its coordinates. This operation is achieved by providing all robots with the desired action plan. The plan indicates what is the role of every robot depending on its coordinates and how the roles change over time. While all the robots know the full action plan in advance, each robot only knows its role once it computes its coordinates through routine R2. Using the synchronisation method described in Sect. 3.1, the robots can also synchronously move to the next step of the action plan. Such a synchronous role change can be interpreted as a change in the swarm state. Enabling swarm state changes allows the programming of robot swarms through swarm-level finite state machines, which can simplify the design of collective behaviour, as was also suggested by previous research (Pinciroli & Beltrame, 2016). Indeed, designing swarm robotics algorithms is complicated as the collective behaviour must be encoded in individual robot rules and the link between swarm and robots can be counter-intuitive. Having the possibility of defining the swarm action plan, while maintaining a fully decentralised approach, can be useful.

In this study, we implemented two types of action plans that differ in how the robots activate. In both action plans, a subset of robots in predefined locations becomes active. In the first case, the active robots activate their motors and move out from the formation. In the second case, the active robots light on their coloured LED and create a collective pattern as shown in Sect. 4. In the second action plan, robot activation is periodically alternated between a cycle of robot subsets with potentially different active roles (implemented as different light colours). The repeated patterns allowed us to test the robustness of the system (e.g., to maintain swarm-level synchronisation) during several-minute-long runs.

3.5 Complexity analysis

The complexity of our routines can be measured by computing the number of messages that each robot is expected to receive and process. Indeed, our subroutines (see Algorithms 1–4) have a single loop that repeats the execution of the listed operations for each received message.

The operations of every subroutine in R1 are local, with each robot only receiving and processing messages from its neighbours and neighbours of neighbours. This number depends on the specific regular formation and robot's communication range, for example, in our experiments with Kilobots in a rectangular lattice, the number of local neighbourhood messages was in the worst case 25. When using simple devices, messages can get frequently lost, hence we let the robots repeat each message exchange for predefined time periods (configured through temporal timeouts). Therefore, the algorithm complexity of routine R1 remains constant and independent of the swarm size N, $\mathcal{O}(1)$, however it increases linearly with longer timeout periods.

Differently, the number of messages exchanged in the subroutines of R2 depends on the swarm size N. In particular, for a rectangular lattice $n \times m = N$ (with $n \geq m$), the complexity of subroutines SR2a and SR2c scales as the diagonal of the rectangle, which in discrete space corresponds to the longer edge n. In subroutine SR2b, the complexity scales as the rectangle diameter. Therefore, for rectangular lattices, routine R2's complexity is $\mathcal{O}(n)$, and for a square lattice is $\mathcal{O}(\sqrt{N})$.

Routine R3 is based on the synchronisation procedure which can be triggered anywhere and concurrently in multiple locations of the swarm. With relatively homogeneous internal clocks, the message exchange remains local, however in the worst case, synch messages need to be spread to everyone scaling as the diagonal of the rectangle, $\mathcal{O}(n)$, or $\mathcal{O}(\sqrt{N})$ for square lattices.

4 Experiments

We tested the algorithms through a series of experiments in simulation and with swarms of real robots. As indicated earlier, the chosen robotic platform is the Kilobot robot (Rubenstein et al., 2012, 2014a), which is a relatively simple robot that can move using vibration motors and exchange IR messages with other robots in a range of about 10 cm. The Kilobots can also estimate the distance (but not the position) of the sender of any received message. Finally, they can show their internal state through a coloured light-emitting diode (LED).

The code to run our algorithm on the Kilobots is open source and available both on Github at https://github.com/TBU-AILab/Kilobot-self-localization and in the Zenodo repository https://doi.org/10.5281/zenodo.14599973.

4.1 Simulations

The simulations have been performed with ARGoS (Pinciroli et al., 2012), a modular and efficient simulator for swarm robotics (Pitonakova et al., 2018), which provides a convenient plugin for simulating the Kilobots (Pinciroli et al., 2018). The Kilobot-plugin for ARGoS allows the user to implement robot code that can be transferred directly from simulation to physical robots without modification, thereby facilitating both implementation and testing.

All simulations were run with a time step of 0.1 s, using synchronous communication among robots. No noise was introduced in sensing and communication, allowing us to focus on evaluating the algorithm's core logic and behaviour under ideal conditions.

Through simulation, we performed several tests, running numerous independent repetitions with different random seeds for each investigated condition. We conducted tests in experiments with up to 1000 simulated Kilobots, where we tested rectangular lattices of different dimensions, from 3×3 to 40×25, and in each case, we varied the inter-robot spacing, from 35 to 70 mm. In all experiments, the swarm successfully completed the process: every robot correctly computed its coordinates and took on its expected role. Figure 5 shows four screenshots of a simulation with 10×10 Kilobots illustrating the four main phases of the algorithm: neighbourhood construction (Fig. 5a), axes' origin selection and border counting (Fig. 5b), and assignment of the x-axis values (Fig. 5c) and of the y-axis values (Fig. 5d).

4.2 Kilobot experiments

We also conducted a range of real-robot experiments employing up to 200 Kilobots. We tested different grid sizes, topologies, and inter-robot spacing, as illustrated in Figures 1, 6, 7, and 8. Experiments have been conducted in three different laboratories. Small-scale experiments, with up to 25 Kilobots, have been conducted in the Swarm Lab at the New Jersey Institute of Technology, large-scale experiments with 64–200 robots have been conducted using the Kilobot infrastructure of Sheffield Robotics (Nikolaidis et al., 2017) at the University of Sheffield, and the experiments to quantify speed and robustness of the system have been conducted in the A.I.Lab of the Tomas Bata University in Zlín, Czech Republic.

Scalability. Kilobot experiments tested the capability of the algorithm to run without change in swarms of different sizes. We conducted experiments with square lattices of 5×5 robots (Fig. 6), 8×8 robots, 10×10 robots (Fig. 7), and 25×8 robots (Fig. 8). The figures show key frames of the process, always terminating with the dynamic role assignment in the form of light patterns that show written text. Note that the axes' origin and orientation are determined at run time and are randomly chosen, therefore in about half of our experiments the text appeared mirrored (when the axes' orientation is inverted the displayed text is subject to a reflection). For ease of visualisation, we only report images and videos of runs in which the light pattern is displayed in the same orientation as the observer.

In Fig. 6, the 25 robots have a cyclic role assignment in which they form the light pattern N-J-I-T. The video of one experiment with 25 robots is available at https://youtu.be/RdUs_EHRnUU and in the supplementary online material (Video 1). Figure 7 shows frames for

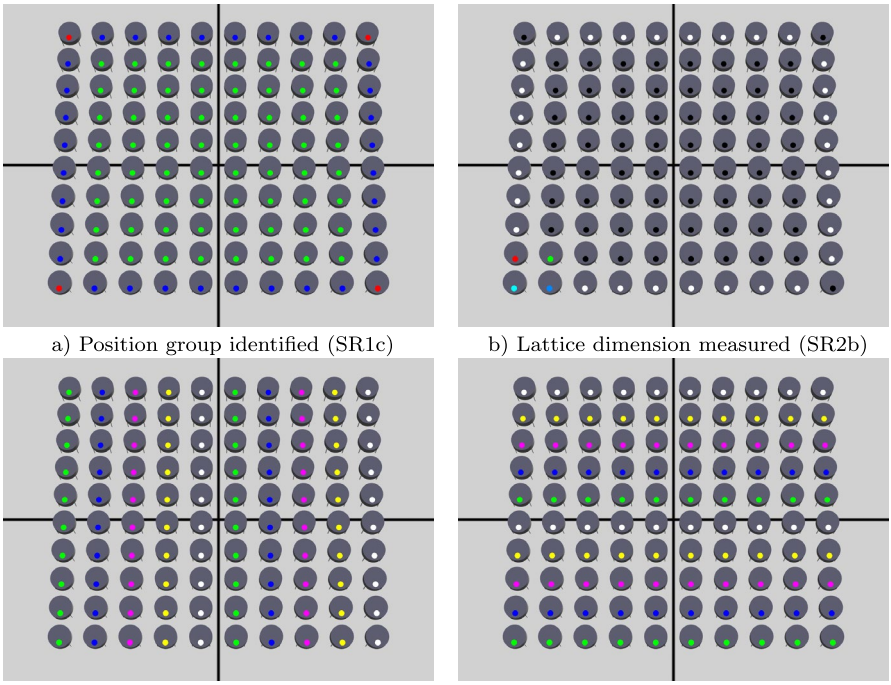


Fig. 5 Four key screenshots from an ARGoS simulation with 100 Kilobots. Panel **a** shows the completion of subroutines SR1c where robots in the three position groups CORNER, BORDER, and MIDDLE light up with colours red, blue, and green, respectively; thus, reproducing the results obtained with real Kilobots in Fig. 1a. Panel **b** shows the completion of subroutine SR2b with the axes' origin in the bottom left corner, and the robots on the borders that have received the border-count message have their white light on, analogous to Fig. 4. In panels **c** and **d**, the robots self-assigned roles (routine R3) depending on their x-axis and y-axis coordinates, respectively. The coordinate-based colours are only based on five colours, which therefore are repeated twice in the 10×5 lattice

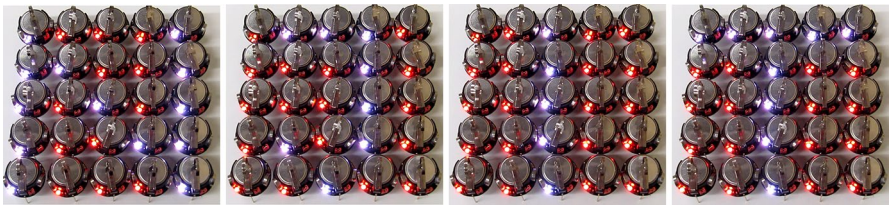


Fig. 6 25 Kilobots in a square 5×5 lattice self-assign roles in synchrony with each other. They form a repetitive cycle of light patterns displaying the letters N-J-I-T

a swarm of 100 Kilobots in some of the key algorithm phases which terminate with the cyclic role assignment of two light patterns forming the worlds "HE-LLO" and "WORLD". The video of one of the experiments with 100 robots is available at https://youtu.be/KlooXOOvZsY and in the supplementary online material (Video 2). Finally, Figure 8 shows frames from experiments with the largest swarm tested, organised in a rectangular

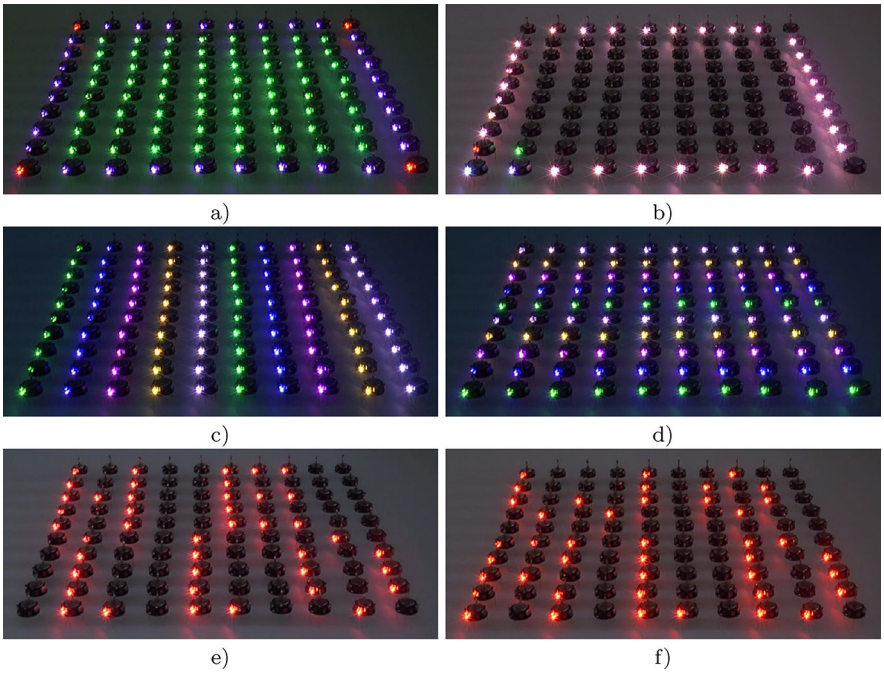


Fig. 7 Six screenshots of an experiment with 100 Kilobots in a square 10×10 lattice. Panels **a**–**d** display the same four moments as displayed in Fig. 5 with simulated robots. Panels **e** and **f** show two moments of the synchronised dynamic role assignment (routine R3), where the robots form a repetitive cycle of light patterns displaying the words "HE-LLO" and "WO-RLD", respectively. The video of the full experiment is available at https://youtu.be/KlooXOOvZsY and in the supplementary online material (Video 2)

lattice sized 25×8. In these experiments, the cyclic role assignment displays the word "SWARM" that alternates between two different positions in the lattice. The video of one of the experiments with 200 robots is available at https://youtu.be/S4s6fpWZvMM and in the supplementary online material (Video 3).

Through a series of 80 robot experiments with up to 100 robots, we also tested the speed to complete the routines R1 and R2 and the various subroutines within. Figure 9 shows the results of the 80 experiments, where we run 10 repetitions for each of the eight tested lattice sizes, $3 \times 3, 4 \times 4, \ldots, 10 \times 10$. If we consider that the the horizontal axis is in logarithmic scale (as at each tick the swarm size has a quadratic increase), the scaling of the algorithm in terms of competition time is relatively good. The speed of the process can be potentially further optimised by tailoring the time limits of the subroutines to the specific scenario. Anyhow, our experiments using the time limits indicated in Sect. 3 already show a relatively quick process compared with other research experiments that used large-scale swarms of Kilobots (Rubenstein et al., 2014b; Reina et al., 2017, 2018; Gauci et al., 2018).

Lattice topologies and inter-robot spacing. Through a set of Kilobot experiments, we tested different topologies and spacing among robots. While routine R1 can work on a variety of different regular lattices (as discussed in Sect. 3.2), routine R2 has been designed for rectangular lattices only. Figure 1 shows the final stage of routine R1 in the square and hexagonal lattices, in which the robots display with distinct colours their position group (i.e., CORNER, BORDER, or MIDDLE). Figures 6 and 7 show square

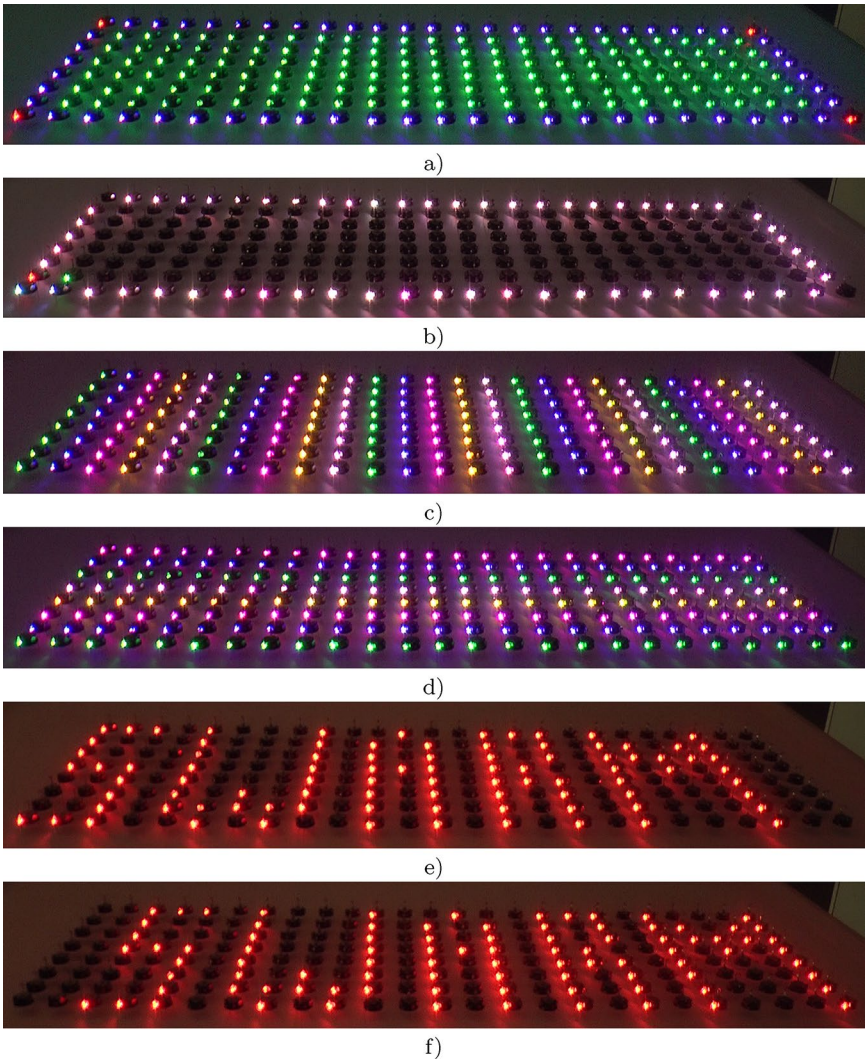


Fig. 8 Six screenshots of an experiment with 200 Kilobots. Panels **a**–**d** display the same four moments as displayed in Figs. 5 and 7 in smaller groups. Panels **e** and **f** show two moments of the synchronised dynamic role assignment (routine R3), where the robots form a repetitive cycle of light patterns displaying the word "SWARM" shifted by two columns. The video of the full experiment is available at https://youtu.be/S4s6fpWZvMM and in the supplementary online material (Video 3)

formation with largely different spacing among robots. Finally, in the experiments of Fig. 8, we tested a rectangular lattice (non-squared). In addition, these tests also validate the robustness of our algorithm to misplacement errors which have been introduced by the manual placing of the robots in the lattice formation. For example, in Fig. 7 it is possible to appreciate the visible misalignment of some of the robots on the second column from the right, which are particularly evident when the robots have their lights turned on.

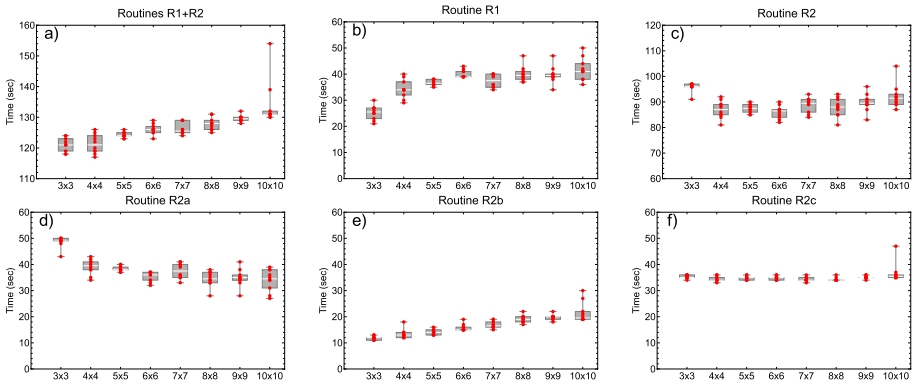


Fig. 9 Temporal scaling in the execution of Routines R1 and R2. The plots show the aggregated data (boxplot) and individual datapoints (red markers) of 10 robot experiments for each grid size (indicated on the x-axis). The text on the top of each plot indicates the measured time. Note that the swarm size on the horizontal axis is in logarithmic scale. The competition time of scales well with increasing swarm size and the variance is relatively low, except a few occasional runs

Robustness to robot failures. We study the robustness of the system to individual robot failures. In particular, we consider the impact of the breakdown of one or more robots which become irresponsive and stop communicating with their neighbours. We simulate the breakdown by physically removing the robot from the lattice. Depending on the moment of the breakdown and the location of the broken robot, the impact on the system is different.

A failing robot can compromise the execution of routines R1 and R2 when it is located in a corner or on the border of the lattice. Robots in these locations play critical roles in the information gossiping through the swarm. In contrast, as discussed below, failures of robots in the middle of the lattice typically have limited or no impact on the successful execution of the three routines. Assuming p_b to be the breakdown probability per robot, we can quantify the probability of a catastrophic event (i.e., full system failure) as the probability that a robot in the corner or edge of the lattice breaks. This probability increases with the swarm size N. For example, assuming a square lattice with layout $\sqrt{N} \times \sqrt{N}$, the system failure probability is

$$1 - (1 - p_b)^{4(-1+\sqrt{N})}. \tag{2}$$

In Fig. 10(a), we report the scaling of this curve for four values of $p_b \in \{0.01, 0.005, 0.001, 0.0001\}$. When p_b is relatively high, e.g., $p_b = 1\%$ of the robots are likely to fail, large systems have a high probability of not functioning. For instance, for such a high value of p_b, the probability of whole system failure exceeds 50% when the swarm size is greater than approximately 333 robots. Therefore, additional information recovery routines should be considered before deploying the algorithm to such contexts. Instead, when the probability that a robot fails in p_b is low (e.g., $p_b = 0.01\%$), also large swarms (e.g., $N = 10^4$) are likely to complete the process without fatal disruptions.

The predictions of Fig. 10(a) are only valid when the breakdown happens before the end of routine R2. Failures that happen after completion of routine R2, i.e., when the robots computed their coordinates, do not have any impact on the other robots during the routine

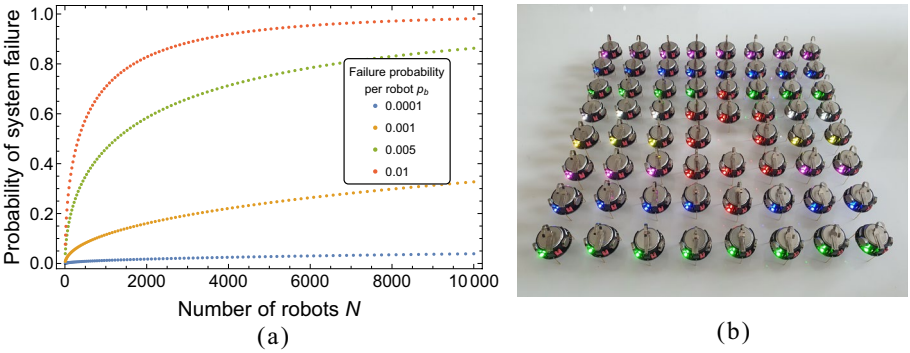


Fig. 10 **a** The current system is vulnerable to the breaking of robots located in the corners or the edge of the lattice. Given different individual probabilities of a robot breaking down (i.e., becoming inactive), the probability that one of them is in one of the critical locations (corner, edge) increases with the swarm size N (on the x-axis). When robots are prone to failure, implementing recovery mechanisms on corners and edges is critical to enable system scaling. Computed using Eq. (2) for square lattices $\sqrt{N} \times \sqrt{N}$. **b** Failures in middle robots only cause a localized error in the lattice, in the photo the robots lighted in red computed incorrect coordinates. All robots can compute the correct coordinates when one or more middle robots fail after R1, see Video 4 in the supplementary online material

R3. Hence, our analysis is only relevant to failures happening before the beginning of routine R1 or after the execution of R1 but before the start of R2. As reported in Fig. 9, the execution of R1 and R2 is particularly quick, thus limiting the likelihood of breakdowns during their execution.

When the breakdown is prior to the start of routine R1 (i.e., at the beginning of the process), the robots adjacent to a failing robot located in the middle of the lattice compute their relative position incorrectly, as they have fewer neighbours than they should. This type of failure is visualised in Fig. 10(b), where we report colour-coded results of the coordinate assignment at the end of routine R2. As Fig. 10(b) shows, the breakdown of one middle robot negatively impacts the robots in its proximity which incorrectly compute their coordinates, yet the disruption remains localised and does not compromise the entire grid. When the breakdown of one or two middle robots happens after the completion of R1 and before the beginning of R2, it has no effect on the other robots. We run a set of robot experiments to test these conditions; see Video 4 in the supplementary online material. However, when more than two adjacent robots break down, depending on their spatial configuration, they can affect the computation of other robots in their neighbourhood. If three failed robots form an L-shape or lie along a diagonal (see Video 4 in the supplementary online material), the rest of the swarm remains unaffected. In contrast, when three adjacent robots in a straight line break down, two neighbouring robots incorrectly compute one of their coordinates.

In summary, while the system is resilient to failures of individual robots located in central regions or occurring after the execution of routine R2, it remains vulnerable to early failures in critical positions—particularly at the corners and edges of the lattice. Therefore, in applications involving large swarms or environments with a non-negligible risk of hardware failure, the algorithm may require the integration of redundancy or recovery routines to maintain global functionality.

Dynamic role assignment. In Figures 6, 7, and 8, the role assignment consisted of displaying a repetitive cycle of light patterns. We let the swarm run these light patterns for several dozen minutes, and in all tested cases, the swarm never desynchronised. In an additional experiment, we tested a different role assignment scenario in which a subset of Kilobots at given locations commenced movement and left the formation. The video of this experiment is available at https://youtu.be/4wlstUNpNcU and in the supplementary online material (Video 5).

Summary. The robot experiments confirmed the validity of the proposed algorithm by consistently reaching the successful completion of all routines. In our scalability tests, we showed that the same identical algorithm can be employed for small grids as well as for large-scale experiments, in our case up to 200 real robots. In this section, we presented more than 100 experiments with real robots which always led to the correct assignment of the coordinates of every robot, except for a few cases where we intentionally deactivated some of the robots to simulate faults in critical positions. The speed measurements showed good scalability performance with the system size. The final routine of cyclic role assignment has also been instrumental in validating the possibility of changing the collective state of the swarm in a synchronised way, even in large swarms composed of robots that only use local communication. Additionally, real robot experiments tested the ability of the algorithm to operate under variable levels of communication noise and distance measurement errors, intrinsic to such simple devices. Tests with different topologies and with robot formations with some placement inaccuracies due to manual setup further showed the generality of the proposed approach and its robustness. The system can sustain different types of robot breakdowns however failure of certain robots in specific locations and at specific times can compromise the entire system process. Our analysis and experiments quantify these conditions and the probability of them to occur.

5 Conclusion

We proposed a decentralised algorithm that enables stationary robot swarms, organised in a regular lattice, to collectively build a shared reference system through which every robot can self-localise within the group. This spatial information allows robots to compute their unique coordinates within the lattice and self-assign roles based on their position. Enabling positional self-awareness and location-dependent task allocation in stationary robot swarms (e.g., those deployed in formation at startup) supports the execution of higher-level collective behaviours, such as coordinated motion in formation (Pratissoli et al., 2023) or localised computation (Beal and Viroli, 2015). Our algorithm also enables temporal coordination of robots' actions through a combination of open- and closed-loop synchronisation mechanisms. Synchronised behaviour is key to achieving effective coordination, both in group-living organisms (Couzin, 2018) as in artificial swarms (Trianni & Nolfi, 2009; Ghosh et al., 2022). We showcase our solution in a set of experiments comprising up to 200 real robots that create synchronised dynamic light patterns.

The main distinguishing aspect of our approach compared to previous solutions is the possibility of constructing a shared swarm-level coordinate system with swarms of minimalistic robots, which are unable to measure the bearing of other robots. Our algorithms can run on robots capable only of basic computation, broadcasting small messages, and making noise-prone estimations of the distance of nearby robots. In contrast, most previous

work could only achieve decentralised self-localisation through more complex robots equipped with range-and-bearing sensors (Beal et al., 2013; Sahin et al., 2002; Guo et al., 2011; Coppola et al., 2019; Mathews et al., 2017; Wang & Rubenstein, 2021; Batra et al., 2022; Li et al., 2018; Klingner et al., 2019). Algorithms for minimalistic platforms have higher portability, due to fewer robot requirements, and can, therefore, be used in a wider range of applications. We implemented our algorithm on swarms of up to 200 real Kilobot robots, a reference platform for minimalistic collective robotics.

While our algorithm does not require the ability to detect the bearing of incoming messages, it assumes the robot swarm is initially deployed in formation, creating a regular lattice (e.g., a rectangular lattice). The algorithm can thus exploit the regularity of the lattice and its geometric properties to compute the relative location and bearing of each neighbour. This assumption is generally not needed in the previously cited algorithms that rely on robots equipped with range-and-bearing sensing. Our solution trades robot simplicity for the need for a predefined spatial arrangement; as discussed in Sect. 3.2.3, the robots do not need to know their formation a priori, but can instead deduce the lattice topology online through subroutine SR1c, distinguishing between rectangular and hexagonal lattices. The deployment of large-scale swarms in regular lattices is motivated by the customary practices of storing, charging, and transporting robots organised in such regular formations. This constraint is also in line with previous studies on collective robotics that required the deployment of the robot swarm in a regular lattice (Rubenstein et al., 2014b; Gauci et al., 2018; Slavkov et al., 2018; Pratissoli et al., 2019, 2023). Some of these works also ran self-localisation algorithms on Kilobots without using bearing information on messages; however, unlike our approach, they required the use of a small set of robots with specific locations and algorithms to act as 'coordinate seeds' (Rubenstein et al., 2014b; Gauci et al., 2018). A further advantage of our algorithm is the absence of predefined roles; instead, robots are able to self-determine their role and position at runtime. All robots run the same code and can be freely interchanged. While the current algorithm is limited to a few regular lattices, future research could explore how our approach might be extended to other regular lattices, to irregular formations, and to higher dimensions. For example, we anticipate that minor changes could enable our routines to operate in swarms in three dimensions, organised in monoclinic and orthorhombic Bravais lattices, making the framework applicable and relevant to the recent 3D modular self-assembling robotic platforms e.g., Nisser et al. (2022), Bray & Groß (2023).

This study takes a practical approach, demonstrating the feasibility of running our algorithm on Kilobot swarms across a variety of conditions, lattice formations, and swarm sizes. Experimental results confirm that the algorithm performs reliably on simple, error-prone robotic platforms. To improve robustness, we incorporated several mechanisms to compensate for individual errors and enable the swarm to reach global coordination. However, future work should aim to enhance the algorithm's resiliency by incorporating mechanisms that prevent collective failure when robots fail at critical locations within the lattice (e.g., those at the coordinates' origin or along the boundaries). While we have demonstrated that the swarm remains functional despite random failures of robots in the centre of the formation, our results show that failures at the border or corners can impair system performance. Developing strategies to detect and compensate for such failures is essential to strengthening system robustness, eliminating single points of failure, and enabling higher levels of collective fault tolerance. Another promising direction for future research is to apply the algorithm to swarms of robots moving in regular lattice formations, such as robotic sheets composed of self-propelled autonomous modules, as explored in (Pratissoli et al., 2023). Ultimately, enabling reliable spatiotemporal awareness in minimalistic

swarms provides a foundational capability on which more advanced, self-organised and coordinated swarm behaviours can be built.

Supplementary Information The supplementary online material with all videos of our robot experiments is available at https://doi.org/10.1007/s11721-025-00251-4.

Acknowledgements The authors thank James A.R. Marshall for granting access to Sheffield Robotics's infrastructure for Kilobot experimentation and Michael Port for technical assistance during the experiments.

Author contributions All authors conceived the original idea. M.P. designed the algorithm and conducted the experiments. All authors analysed the results and wrote the article.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by OP VVV project International Mobility of Researchers of TBU in Zlín project no. CZ.02.2.69/0.0/0.0/16_027/0008464 and further by program "Excellence initiative—research university" for the AGH University in Krakow as well as the ARTIQ project: UMO-2021/01/2/ST6/00004 and ARTIQ/0004/2021. A. Reina acknowledges support from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2117–422037984. S. Garnier acknowledges support from the National Science Foundation under grant no. #EF-2222418.

Data availibility The code to run our algorithm on the Kilobots is open source and available at https://doi.org/10.5281/zenodo.14599973 and on Github at https://github.com/TBU-AILab/Kilobot-self-localization.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Consent for publication The authors give their consent for publication.

Ethical approval Not applicable.

References

Alhafnawi, M., Hauert, S., & O'Dowd, P. (2020). Robotic canvas: Interactive painting onto robot swarms. *Artificial Life Conference Proceedings, 32*, 163–170.

Barciś, A., & Bettstetter, C. (2020). Sandsbots: Robots that sync and swarm. *IEEE Access, 8*, 218752–218764.

Batra, S., Klingner, J., & Correll, N. (2022). Augmented reality for human-swarm interaction in a swarm-robotic chemistry simulation. *Artificial Life and Robotics, 27*(2), 407–415. https://doi.org/10.1007/s10015-022-00763-w

Beal, J., & Viroli, M. (2015). Space-time programming. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 373*(2046), 20140220.

Beal, J., Dulman, S., Usbeck, K., Viroli, M., & Correll, N. (2013). Organizing the aggregate: Languages for spatial computing. In: *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments,* IGI Global, pp 436–501.

Bray, E., & Groß, R. (2023). Recent developments in self-assembling multi-robot systems. *Current Robotics Reports, 4*(4), 101–116. https://doi.org/10.1007/s43154-023-00106-y

Coppola, M., Guo, J., Gill, E., & de Croon, G. C. (2019). Provable self-organizing pattern formation by a swarm of robots with limited knowledge. *Swarm Intelligence, 13*(1), 59–94.
Couzin, I. D. (2018). Synchronization: The key to effective communication in animal collectives. *Trends in Cognitive Sciences, 22*(10), 844–846.
Ganesh, A. J., Kermarrec, A. M., Le Merrer, E., & Massoulié, L. (2007). Peer counting and sampling in overlay networks based on random walks. *Distributed Computing, 20*(4), 267–278.
Gauci, M., Nagpal, R., & Rubenstein, M. (2018). Programmable self-disassembly for shape formation in large-scale robot collectives. *Distributed Autonomous Robotic Systems (DARS 2016): The 13th International Symposium, SPAR* (Vol. 6, pp. 573–586). Cham, Switzerland: Springer.
Ghosh, D., Frasca, M., Rizzo, A., Majhi, S., Rakshit, S., Alfaro-Bittner, K., & Boccaletti, S. (2022). The synchronized dynamics of time-varying networks. *Physics Reports, 949*, 1–63.
Greunen van, J., & Rabaey, J. (2003). Lightweight time synchronization for sensor networks. In: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications, Association for Computing Machinery, New York, NY, USA, WSNA '03, p 11–19.
Guo, H., Meng, Y., & Jin, Y. (2011). Swarm robot pattern formation using a morphogenetic multi-cellular based self-organizing algorithm. In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp 3205–3210.
Hamann, H. (2018). *Swarm Robotics: A Formal Approach*. Cham, Switzerland: Springer International Publishing.
Jones, S., & Hauert, S. (2025). Distributed spatial awareness for robot swarms. In: *Distributed Autonomous Robotic Systems (DARS 2024): The 17th International Symposium*, SPAR, Springer, Cham, Switzerland, in press.
Klingner, J., Ahmed, N., & Correll, N. (2019). Fault-tolerant covariance intersection for localizing robot swarms. *Robotics and Autonomous Systems, 122*, Article 103306. https://doi.org/10.1016/j.robot.2019.103306
Li, Q., & Rus, D. (2006). Global clock synchronization in sensor networks. *IEEE Transactions on Computers, 55*(2), 214–226. https://doi.org/10.1109/TC.2006.25
Li, Y., Klingner, J., & Correll, N. (2018). Distributed camouflage for swarm robotics and smart materials. *Autonomous Robots, 42*(8), 1635–1650. https://doi.org/10.1007/s10514-018-9717-6
Mathews, N., Christensen, A. L., O'Grady, R., Mondada, F., & Dorigo, M. (2017). Mergeable nervous systems for robots. *Nature Communications* 8(1).
Nagpal, R., Shrobe, H., & Bachrach, J. (2003). Organizing a global coordinate system from local information on an ad hoc sensor network. In: *Information Processing in Sensor Networks (IPSN 2003)*, LNCS, 2634, Springer, Berlin, Heidelberg, pp 333–348.
Naz, A., Piranda, B., Goldstein, S. C., & Bourgeois, J. (2016). A time synchronization protocol for modular robots. In: *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, IEEE, pp 109–118.
Nikolaidis, E., Sabo, C., Marshal, J. A. R., & Reina, A. (2017). *Characterisation and upgrade of the communication between overhead controllers and Kilobots*. Tech Rep: White Rose Research Online.
Nisser, M., Cheng, L., Makaram, Y., Suzuki, R., & Mueller, S. (2022). ElectroVoxel: Electromagnetically actuated pivoting for scalable modular self-reconfigurable robots. In: *2022 International Conference on Robotics and Automation (ICRA)*, pp 4254–4260.
Pinciroli, C., & Beltrame, G. (2016). Swarm-oriented programming of distributed robot networks. *IEEE Computer, 49*(12), 32–41.
Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., & Dorigo, M. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence, 6*(4), 271–295.
Pinciroli, C., Talamali, M. S., Reina, A., Marshall, J. A. R., Trianni, V. (2018). Simulating Kilobots within ARGoS: Models and experimental validation. In: *International Conference on Swarm Intelligence (ANTS)*, Springer, LNCS 11172, pp 176–187.
Pitonakova, L., Giuliani, M., Pipe, A., & Winfield, A. (2018). Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators. In: *Towards Autonomous Robotic Systems (TAROS),* Springer International Publishing, Cham, LNCS, 10965, pp 357–368.
Pratissoli, F., Reina, A., Kaszubowski Lopes, Y., Sabattini, L., & Gross, R. (2019). A soft-bodied modular reconfigurable robotic system composed of interconnected Kilobots. In: *Proceedings of the 2019 IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS 2019)*, IEEE, pp 50–52.
Pratissoli, F., Reina, A., Kaszubowski Lopes, Y., Pinciroli, C., Miyauchi, G., Sabattini, L., & Gross, R. (2023). Coherent movement of error-prone individuals through mechanical coupling. *Nature Communications, 14* (4063). https://doi.org/10.1038/s41467-023-39660-6
Pritsker, A. A. B. (1984). *Introduction to Simulation and SLAM II*. Halsted Press.

Reina, A., Cope, A. J., Nikolaidis, E., Marshall, J. A. R., & Sabo, C. (2017). ARK: Augmented reality for Kilobots. *IEEE Robotics and Automation Letters, 2*(3), 1755–1761.

Reina, A., Bose, T., Trianni, V., & Marshall, J. A. R. (2018). Effects of spatiality on value-sensitive decisions made by robot swarms. In: *Distributed Autonomous Robotic Systems (DARS 2016): The 13th International Symposium,* SPAR, vol 6, Springer, Cham, Switzerland, pp 461–473.

Rubenstein, M., Ahler, C., & Nagpal, R. (2012). Kilobot: A low cost scalable robot system for collective behaviors. In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp 3293–3298.

Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., & Nagpal, R. (2014). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems, 62*(7), 966–975.

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science, 345*(6198), 795–799.

Saha, A., Marshall, J. A. R., & Reina, A. (2021). Memory and communication efficient algorithm for decentralized counting of nodes in networks. *PLoS ONE, 16*(11), Article e0259736.

Sahin, E., Labella, T. H., Trianni, V., Deneubourg, J. L., Rasse, P., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., & Dorigo, M. (2002). SWARM-BOT: Pattern formation in a swarm of self-assembling mobile robots. *IEEE International Conference on Systems, Man and Cybernetics, IEEE Press, 4*, 1–6.

Sivrikaya, F., & Yener, B. (2004). Time synchronization in sensor networks: A survey. *IEEE Network, 18*(4), 45–50. https://doi.org/10.1109/MNET.2004.1316761

Slavkov, I., Carrillo-Zapata, D., Carranza, N., Diego, X., Jansson, F., Kaandorp, J., Hauert, S., & Sharpe, J. (2018). Morphogenesis in robot swarms. *Science Robotics 3*(25).

Snyder, W. E., Qi, H., & Sander, W. A. (1999). Coordinate system for hexagonal pixels. *Proceedings of SPIE, Medical Imaging, SPIE publications, San Diego, CA, United States, 3661*, 716–727.

Trianni, V., & Nolfi, S. (2009). Self-organizing sync in a robotic swarm: A dynamical system view. *IEEE Transactions on Evolutionary Computation, 13*(4), 722–741.

Tyrrell, A., Auer, G., & Bettstetter, C. (2006). Fireflies as role models for synchronization in ad hoc networks. In: *2006 1st Bio-Inspired Models of Network, Information and Computing Systems*, pp 1–7.

Wang, H., & Rubenstein, M. (2021). Decentralized localization in homogeneous swarms considering real-world non-idealities. *IEEE Robotics and Automation Letters, 6*(4), 6765–6772.

Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. *Computer networks, 52*(12), 2292–2330.