A fully distributed communication-based approach for spatial clustering in robotic swarms

Gianni A. Di Caro, Frederick Ducatelle, and Luca M. Gambardella

Dalle Molle Institute for Artificial Intelligence (IDSIA),
Galleria 1, 6928 Manno-Lugano, Switzerland
{gianni,frederick,luca}@idsia.ch

Abstract. In this work we propose a novel fully distributed approach to endow robots in a swarm with awareness of their relative position with respect to the rest of the swarm. Such spatial awareness can be used to support spatially differentiated task allocation or for pattern formation. In particular, we aim to partition the robots in the swarm in two (or more) distinct and spatially segregated groups. The distributed approach we propose only relies on local wireless communications and is based on a combination of distributed consensus and load balancing.
We propose two metrics to measure the effectiveness of the obtained partitioning and we test the performance and the scalability of our algorithm in extensive simulation experiments. We also validate it in a small set of experiments with real robots.

Keywords: Swarm robotics, spatial awareness, multi-robot systems, group splitting, load balancing, distributed consensus.

1 Introduction

The aim of this work is to endow robots in a swarm with awareness of their relative position with respect to the rest of the swarm. Such *spatial awareness* can be used to support spatially differentiated task allocation (e.g., split the swarm in different, spatially close, groups, and let each group engage in a different task, such as exploring different regions of an environment), or for pattern formation. The task we first focus on is to assign the robots of the swarm to two different classes, C_0 and C_1, in such a way that *the two classes are spatially segregated*: the robots in class C_0 are found on one side of the swarm, and the robots in class C_1 on the other side of the swarm, as shown in Figure 1.

To solve this problem, we look for an algorithm that is robust, scalable, efficient, works in a decentralized way, and has limited requirements in terms of available sensor or actuators. A simple solution could be to indicate one or more robots of the swarm as a seeds, and let all other robots calculate their distance to these seeds, e.g. in terms of a communication hop count (similarly to what happens in works on morphogenesis and morphogen gradient [26]). However, this solution requires the explicit indication of the seeds, and is not robust, since it is dependent on the proper functioning of the seed robots. Another solution could

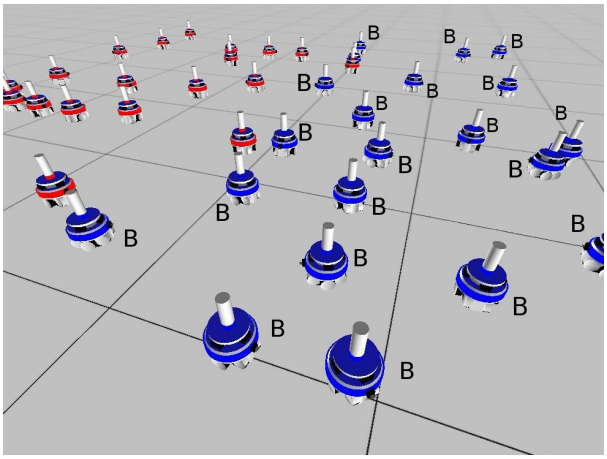

Fig. 1. Example of spatially segregated classes. Robots in class C_0 show blue LEDs (a 'B' label is positioned beside each robot in class C_0 to help readability also in greyscale), robots in class C_1 show red LEDs (and have no text label).

be based on global positioning information and flooding of this information. However, that would require such a position system to be present, which is not always straightforward, especially in indoor environments. Finally, we also exclude approaches where robots move in order to cluster closer to robots of the same class (e.g., using swarm clustering algorithms [10,25]). We do not want to use mobility in the solution, in order to improve efficiency and speed of the system, and to be able to perform the spatial segregation also in cases where robots cannot move for some reason.

Instead, we propose an algorithm which uses only *local communication*. The robots/nodes only need to be able to identify their neighbors and communicate with them. Only a relatively *low bandwidth channel* is required to let the algorithm work effectively. We consider the general case of robots/nodes equipped with a wireless communication interface. The algorithm combines elements from different sets of approaches to similar problems: algorithms for solving *minimum bisection problems* [2,17]; algorithms for *swarm robotics aggregation*, in particular those inspired by cockroach aggregation behavior, in which robots form clusters by moving around and stopping randomly, (with stop probability and duration depending on local group size), [10,25]; distributed algorithms for *consensus load balancing* [8], where instead of cockroaches/robots moving in 2D space, computational loads move in a communication network; and algorithms for *distributed consensus filter*, which create a field the intensity of which depends on local group size (of loaded robots/nodes) [21].

The rest of the paper is organized as follows. In the next section we provide a general overview of the problem we are considering and its relationship with similar problems. In Section 4 the proposed algorithm is described. Experimental results for studies carried out in simulation are presented in Section 5, while

implementation and results using real robots are discussed in Section 6. Finally, in Section 7 we draw conclusions and highlight future work.

2 Problem description

The problem that we are solving can be formalized in the following way. Let $G(V, E)$ be a Euclidean graph where the node set V represents geometric entities, such as robots, positioned in the plane. Nodes are able to communicate with each other over a wireless medium. Two nodes i and j are connected by a link $(i, j) \in E$ if: (i) their Euclidean distance is less than or equal to the maximum communication range R_{max} (*range-constrained connectivity*), and (ii) no major occlusions are present between the two nodes (*line-of-sight communication constraint*). The length of a link (i, j) is equal to the Euclidean distance between i and j. Each node only knows its neighbors, that is, nodes falling within its communication range (local topological information). Information about the other nodes in the system, such as their total number or their global positioning (e.g., GPS information) is assumed as not available. The objective is to find, adopting a fully decentralized approach, a geometric partitioning of the graph in k classes, where each class contains (approximately or precisely) the same number n_k of nodes, and the nodes in each partition are geometrically close to each other. That is, the different partitions are spatially separated, as in shown in Figure 1 for a partitioning in two classes. In the rest of the paper we focus on the case $k = 2$. The method can be naturally extended to handle geometric partitioning for $k > 2$ classes by using k-valued tokens and load variables (see Section 4).

In order to complete the formulation of the problem, a precise way to measure the notion of geometric partitioning is needed. Figures 2(a) and 2(b) show two equally acceptable geometric partitioning in two classes for a given placement of a robotic system and for a given communication range (indicated by the grey disk in Figure 2(a). Figure 2(c) shows an example of incorrect partitioning: not all nodes in the class indicated with the black squares are geometrically close to each other according to their communication range. In this paper, we measure the goodness of a geometric partitioning in terms of *linear separability* and *class imbalance*. Linear separability measures in how far the nodes of the different classes can be separated by a straight line in the two-dimensional space in which they are placed. Class imbalance measures the difference among the number of nodes in each class (the values n_k): we aim to have an equal number of nodes belonging to the k classes.

While reaching a perfect balance is clearly a primary objective in the context of the problem we are considering, linear separability might look as a requirement which is too strong or inappropriate, even for the case $k = 2$ we are focusing on. In fact, depending on the actual deployment of the nodes in the 2D space, it might be unfeasible to reach perfect (or near perfect) linear separability and, at the same time, also a perfect balance. However, since none of the metrics that have been proposed in literature (see next section) seems to satisfactorily match our goal of defining spatially segregated groups, we decided to apply

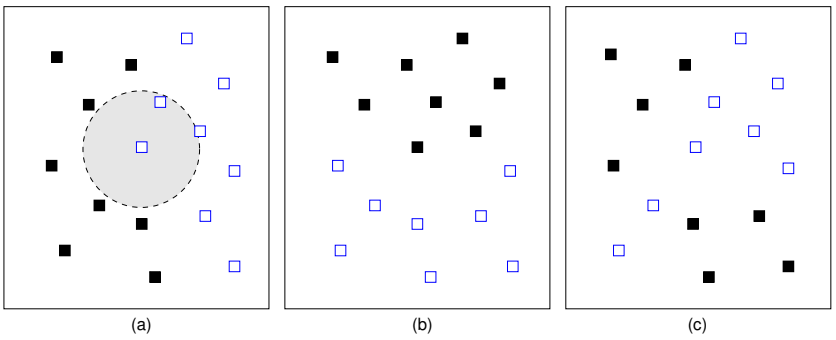

Fig. 2. Examples of different ways to realize a geometric partitioning in two classes (indicated by the black and white squares) given a communication range (indicated by the grey disk in (a)). Sub-figures (a) and (b) shows two different but equally acceptable ways to realize the partitioning. Sub-figure (c) shows an example of incorrect partitioning.

a sort of Occam's razor: the presence of a linear separation, coupled with a good balance, would strongly indicate that we have reached our objective. The advantage of linear separability as a performance metric lies in its simplicity, immediate visual verification, and clear relationship with a 'good' partitioning in two classes. Alternative metrics would likely require more complex calculations (e.g., piece-wise linear separability) and/or specific definitions of measures such as inter-node and intra-partition distance or geometrical spreading (e.g., these type of measures are often adopted, by no means free of problems, in the context of data clustering). It is important to remark that the algorithm that we present in the following is not explicitly designed to obtain linear separability. Such as, using linear separability as metric, we implicitly penalize the performance of our approach.

3 Related work

In the context of the problem description given in the previous section, different ways of formulating the objective, as well as the constraints and the accessible knowledge, result in a number of related different problems that have been studied in different domains. In the following of this section, we briefly review these related formulations and discuss proposed approaches.

The *k-way graph partitioning problem* (k-GPP) aims to find k disjoint partitions $V_1, V_2, \ldots, V_k$ of V such that $\big||V_i| - |V_j|\big| \leq 1 \ \forall i, j \in \{1, \ldots, k\}$, and $\bigcup_{i=1}^{k} V_i = V$. If the two end-nodes of an edge (i, j) belong to different partitions, the edge is called a *cut edge*. The objective is to find the partitioning that minimizes the *cut size* or the *cut cost*, that is, the number or the cost of the cut edges. For $k = 2$, the problem is called the graph bisectioning problem or *bipartitioning problem*. In the *balanced* version of the problem, the constraint on the unit

difference between the cardinalities of the partitions is relaxed. In this case, the objective is to minimize, in addition to the cut size/cost, also the total difference in cardinality among the partitions, which is the same as our imbalance metric. In this case, the problem is also called the *min-k-cut*. All these k-way graph partitioning problems find a number of applications in VLSI circuit placement, parallel computing, and social networking, among others. Unfortunately, in the large majority of the cases of interest, the k-GPP formulation is NP-hard, and a number of approximate and heuristic approaches have been proposed, mostly based on centralized approaches and full system knowledge, such as the coordinates of the nodes and the total number of the nodes in the network (e.g., see [2,18,4,2,17] for surveys and references). While k-GPPs are closely related to our problem, the minimization of cut size/cost is the not enough to guarantee the geometric partitioning we aim to: additional, explicit geometric constraints need to be added to the formulation, making the problem even more difficult to be solved in practice. Figure 3 illustrates this fact with a simple example: the the 2-partition corresponding to the central dashed line that perfectly fits our objectives has a much higher cost than the 2-partition realized by grouping the nodes as indicated by the diagrams in the figure.

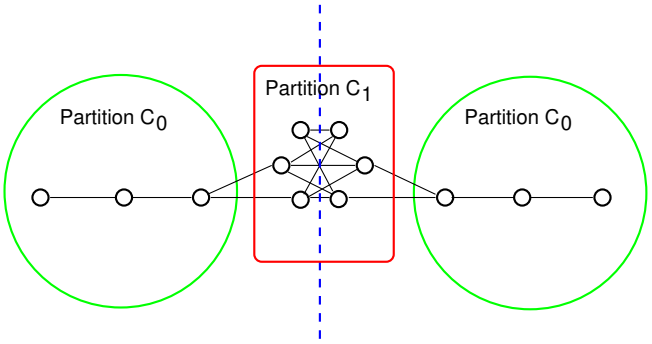

Fig. 3. Illustration of the solution produced by a 2-GPP formulation minimizing cut size. Solution cost is 4. The partitioning indicated by the central dashed line, splitting the nodes in two geometrically well segregated groups has cut size 8. Therefore, this partitioning, which is the kind of partitioning we aim to, would not be selected by a GPP approach.

The problem we consider is also related to *cluster formation in sensor and mobile ad hoc networks*. In these domains, clustering is usually functional to the generation of a hierarchical structure in the network. Each cluster is associated to a cluster-head that communicates with the other nodes in its cluster and acts as a relay node for them. In practice, the cluster-heads form an overlay network that serves to optimize data aggregation and routing. The difference with our formulation mainly consists in the fact in these networks the identification of the clusters and of the cluster-heads is precisely functional to the role that these nodes have to play. The main objective is to find the optimal number of clusters

based on the optimization of metrics such as energy and interference minimization within the cluster, and structure and bandwidth of the routing paths in the overlay [14,9,6,27,19]. Geometric proximity among the nodes in the clusters can result as a by-product of the optimization of these objectives, but it is neither guaranteed nor enforced. Differently from our way of proceeding, distance is also often taken into account, assuming that some global or local positioning system is in place [13,28]. A general survey on clustering in ad hoc networks can be found in [1].

A number of approaches have considered pattern formation, clustering, and aggregation in *swarm robotic* systems. In the large majority of the cases, this is realized through robot mobility and the application of simple behavioral rules and random decisions that allow the emergence of the desired aggregation patterns at the swarm level [12,20,25,10]. In [22] robot group formation is obtained exploiting the supervision of additional robots with a different point of view. The emergence of spatial clustering is usually directed by the presence of external stimuli in the environment, such as light sources [25], odors [11], or objects to be moved and packed/clustered together, like in the so-called puck clustering systems [16,15]. The external stimuli, which are modified over time in consequence of the robot actions, act as stigmergic coordination signals for the robots. Therefore, in many of the proposed approaches in the literature the robots do not engage in any explicit form of communication. Compared to these existing studies, we aim to obtain spatially aware swarm partitioning without using mobility, but relying instead on a purely local communication-based approach, which is in principle faster, consumes less energy, and is of more general applicability. Moreover, we do not need any external signal to direct the partitioning: it is driven internally, by the notion of communication neighborhood. The presence of an external signal (e.g., a human indicating robot seeds, as discussed in the Introduction), could clearly be exploited to accelerate the whole process.

Finally, also the field of *data clustering* addresses problems similar to the one we consider in this paper. However, in this case, the notion of (generalized) distance among the nodes is central to proposed solution approaches. *K-means*, one of the reference algorithms used to attack data clustering problems, is precisely based on the knowledge and use of node distances, which we ruled out in our approach. An example of the way of proceeding in this field for the implementation and analysis of distributed k-means algorithms can be found in [7].

4 Distributed algorithm for spatially aware robot partitioning

As mentioned in the Introduction, the distributed algorithm we propose contains elements from different approaches for network bisectioning, load balancing, and distributed consensus filters. From the work on load balancing, we derived the notion of *load*, which is moved from node to node to obtain the desired balancing. More specifically, in our algorithm, the membership of a robot in the swarm to one of the two different classes C_0 and C_1, is realized by using *tokens* that are

passed on between robots in the network. Tokens can be either *stationary* or *moving*. Each robot can contain maximum one stationary token, and multiple moving tokens. All other robots belong to class C_0, even if they temporarily hold one or more moving tokens. In this way, the distribution of the tokens across the robot network defines the membership of each robot to one of the two classes. By passing on tokens, or holding on to a received token (making it stationary), robots can dynamically change their class membership. In the following, we refer to these tokens as *loads*, to illustrate more effectively the idea of passing on loads between robots. A robot holding one stationary token, that is, belonging to class C_1, is said to be *loaded*, while a robot with no tokens is unloaded. At robot i, the local *load variable* u_i indicates the number of stationary tokens. u_i is a *binary variable*, meaning that one robot can only hold one stationary token at a time.

The core of the algorithm lies in the rules that define when a token/load moves, where it moves to, and when it stops. The movement and aggregation of loads in the robot network can be considered equivalent to the movement of robots in a 2D space, and in this way, our algorithm for load movement takes inspiration from the mentioned robot clustering algorithms. However, making loads moving using the wireless network, can greatly speedup the response of the system and save energy compared to an algorithm based on robot movements.

To be able to guide the movement of loads, we use a second variable, $v_i \in [0, 1]$. It is an estimate of the *local density of stationary loads*: the fraction of robots in the neighborhood holding a stationary load. Considering that our goal is to create two well separated geometric partitions, one with loaded and one with unloaded robots, the idea of the algorithm is to let loads leave robots positioned in areas of low values of v_i (areas with mostly robots of class C_0), and find unloaded robots with high values of v_i. That is, robots of class C_0 that are in areas of mostly robots of class C_1. Taking such a place, the token contributes to aggregate loads together and, in turn, increases the local v_i values more. So we get a *self-reinforcing process* leading to growing clusters of robots. We also make use of an *exploratory* component in the algorithm, in order to let some loads on the edges of large clusters move to explore the area, such that they can find stronger clusters (in terms of density and size) and stay there, until a *single strong cluster* of loaded robots is eventually created and remains stationary (apart some possible small fluctuations at the edges precisely due to the exploratory behavior).

The value of the v_i variables also provides an indirect indication of the *relative position* of robot i with respect to the frontier of the cluster. For instance, v_i values very close to 1 indicates that a robot is well within the cluster of robots of class C_1, while the v_i value of loaded robots progressively decreases towards zero with the approaching of their position to the frontier of class C_0. In this way, a sort of *gradient field* is formed on top of the load distribution, and can be conveniently exploited for control purposes since it provides to the robots additional information in terms of spatial awareness.

4.1 Algorithm description

The algorithm is structured in multiple steps and operational phases that are described in the rest of this subsection.

Initialization. At the start, each robot decides with a probability of 0.5 whether it is loaded or not. Each loaded robot has exactly one token. As discussed above, all loaded robots are of class C_1, and unloaded robots are of class C_0. The internal load variable u_i of each robot i is set accordingly (1 or 0, respectively). The local density variable v_i is initialized to 0.

After the initialization, the robots start to communicate, with two goals: to update the estimate v_i, and to let tokens/loads travel through the swarm, until they stop at different robots.

Updating of load density variables v_i. To update the local v_i variables, the robots use an *average consensus* algorithm. This is based on the *distributed consensus filter* proposed in [21]. At each step, each robot i combines the raw values (u_j) and the estimates (v_j) of itself and its neighbors, to calculate a new estimate u_i according to the following formula:

$$v_i^+ = v_i + \delta\left(\sum_{j \in \mathcal{N}_i} (v_j - v_i) + \sum_{j \in J_i} (u_j - v_i)\right), \tag{1}$$

where v_i^+ is the value of v_i in the next time step, $\mathcal{N}_i$ is the set of neighbors of robot i, J_i is $\mathcal{N}_i \bigcup i$, and δ is a constant, for which $0 < \delta < 1/m$, where m is the maximum node degree in the swarm communication network (in the experiments we used $\delta = 1/n$, where n is the number of robots in the swarm).

Equation (1) leads to a *low-pass filter*, if the observations u_i are all unbiased estimates of a same phenomenon observed by the different nodes/robots. However, here this is not the case, so instead the formula leads to a local averaging of the values of u_i: since each robot i puts into the formula at each time step the difference between his observation u_i and the estimates v_i of himself and of his neighbor robots, the local estimate v_i of each robot is directly influenced by its current local observation u_i. Through the average calculation with the neighbors, this influence spreads, but its influence is less strong farther aways. Therefore, Equation (1) leads to *local averaging*. The formula is also *robust to packet losses* and *adaptive* to changes in u_i, since it always mixes in the new local observations.

The goal of the consensus filtering is to calculate in v_i an estimate of how loaded the robots in the local neighborhood are: how many stationary loads are held by nearby robots. This will then be used by travelling loads to decide when to leave, where to go, and where to stop.

Moving loads between the robots A load can leave a robot and start traveling through the network. The traveling goes in a number of phases.

Phase 0 (Start). It is the initial phase following the first creation or the reception of a load, with the load being stationary in the current robot. A load can leave this phase to start moving. The choice to do so depends on three elements:

- First, a load in a robot i is linked to a waiting counter c_i. This counter is initialized to a constant when the load first arrives at the robot (in simulation we used 40, in the real robots 120). c_i is reduced by 1 each time step that $v_i < 0.5$ (1 time step = 0.1 s), meaning that the local neighborhood of the robot is rather unloaded than loaded. Then, it is better to further unload the neighborhood. Once c_i reaches 0, it is no more reduced, and the load can now leave.
- Once c_i is 0, at each time step the load can leave robot i with a probability of 0.01.
- A last condition is that the robot should have neighboring robots to communicate with; if this is not the case, the load cannot move.

When a load starts moving and leaves the robot, this is considered as unloaded, so u_i is set to 0.

Phase 1 (Steepest ascent). This is the first moving phase, which is a steepest ascent phase. The load is sent to the neighbor j with the highest value v_j. This way, if there is a local cluster of loaded robots, the load will move to the center of it. When the load reaches a local maximum, it stores its value v_j, and makes a random step: it is forwarded to a random neighbor. After that it advances to the next moving phase.

Phase 2 (Steepest descent). This is a steepest descent phase: the load moves to a local minimum of v_j, meaning that it looks for an area which is unloaded. When it reaches a local minimum, it takes again a random step, before moving to the next phase.

Phase 3 (Steepest ascent). It is again a steepest ascent: the load greedily looks for a new loaded area. When it reaches the maximum of v_j, it compares it with the previously obtained maximum (stored at the end of phase 1). If the new maximum is better, the load is assumed to have reached a more loaded area, and moves on to the next phase. If the new maximum is not better, the load can still move to the next phase with a low constant probability (set to 0.1). Otherwise, the load makes a random step and goes to phase 2 again (steepest descent, in search of a new loaded area).

Phase 4 (Slowest descent). This is a slowest descent for the load: it moves from robot to robot to decreasing values of u_i, until it reaches an unloaded robot ($u_j = 0$), where it moves back to phase 0 (we allow maximum 1 load of phase 0 in each robot, where a load in phase 0 means: this robot is of class 1). The idea is that the slow descent will make the loads rather go towards areas where there are only a few unloaded nodes, so that the load goes to fill small empty pockets. If no unloaded robot is found before reaching a local minimum, the load takes a random step, and returns to phase 1: start all over again.

Once the load has reached phase 0 at a robot i, it sets the local value u_i to 1. Note that while a load is traveling, it never sets the value u_j of the nodes j where it passes to 1, meaning that the presence of a moving load at a node does not make this node loaded. The load entering phase 0 also sets the local waiting counter c_i again to the initial constant value. This means that the load will stay at this node i for a while, allowing the system as a whole to adapt, and the local cluster of loaded robots to grow possibly, before the load can travel again. If more loads cluster around the current load's robot, the local value v_i will get above 0.5, and c_i will not be decrease, keeping the load stationary at its current robot, and letting the cluster grow further.

We point out that in robots with less neighbors, it is easier to get an extreme (high or low, depending on u_i) value for v_i, because there are less neighbors to disturb the local robot average. That is why the algorithm prefers to form clusters of loaded/unloaded nodes on the edges of the swarm, rather than in the center, such that the cluster formation leads easily to two approximately linearly separable groups.

4.2 Scalability of communications

Scalability of communications was one of the constraints that we set to guide the design of the algorithm. In the distributed protocol implementing the above algorithm, communications only happen in the form of local broadcast messages that are periodically transmitted to update v_i values and/or to move tokens. This purely local and asynchronous form of transmission, coupled with the small size of each broadcast message, can guarantee the scalability of the protocol in terms of communications. The structure of a communication packet is as follow:

- 1 byte for message type,
- 1 bit for the value of u_i (more bits are needed if more than two classes are considered),
- 1 bit saying whether the packet contains a load or not,
- 1 byte encoding v_i,
- 3 bits to indicate the moving phase (1-4),
- 5 bits to indicate the id of the load (only used in case there are lossy links, to be able to send acknowledgements, as discussed in Section 6 where the real robot implementation is presented),
- 3 bytes to store the quality of the last cluster (stored at the end of phase 4),
- 1 byte to indicate the recipient neighbor of the packet.

Overall, a communication packet amounts to only 44 bits, that can be conveniently packed in 6 bytes. This allows the protocol to scale well and work effectively also when the wireless channel has very limited bandwidth.

5 Experimental results

We run extensive experiments in simulation using the *ARGoS* simulator [23], a physics-based simulator for heterogeneous multi-robot systems developed during

the *Swarmanoid* project [5] (`http://www.swarmanoid.org`). As reference robot we consider the *foot-bot*, developed during the same project (as such, ARGoS contains reliable physics models of this robot). The foot-bot has a diameter of about 15 cm and it is about 20 cm high. It moves on the ground using a combination of tracks and wheels, for increased stability. It is quite a powerful robot, carrying various sensors and actuators, including two cameras, a rotating distance scanner, a gripper, etc. For the work presented here, the relevant device is the infrared-based *range-and-bearing* module (IrRB) [24,3], that provides local *line-of-sight* communication. It sends messages of 10 bytes, and has a capacity of 10 messages per second (so robots can broadcast an update every 0.1 s). Its maximum range can be of more than 5 m, but in the following experiments it was limited to smaller values in order to be able to do tests in relatively small environments and to test the effect of the range on the algorithm. The IrRB, also provides a measure of the distance and relative bearing between two robots, which was not used in this work. In addition to the simulation results presented in this section, a validation is also provided using a small swarm of real foot-bots (discussed in Section 6).

Each simulation test (50 trials per test) runs for a fixed amount of time (1200 time steps = 2 minutes). We evaluate the solution found by the swarm after this time. We measure two things: *linear separability* and *imbalance*. Linear separability is evaluated by solving an integer linear program (ILP) that fits a line to the space in which the robots are placed, in such a way that the loaded robots are found on one side of the line and the unloaded ones on the other side. The result of the evaluation is the total number of robots placed on the wrong side of the line divided by the total number of robots in the swarm. *If linear separability is optimal, the result is 0, while the worst possible score is 0.5.*

The ILP for the evaluation of linear separability is:

$$
\begin{aligned}
&\min \textstyle\sum_{i=1}^{n} r_i \\
&\text{s.t. } M r_i + w_i x_i a + w_i y_i b + w_i c \ \geq 0, \ i = 1, \ldots, n \\
&\qquad a + b + 2k = 1 \\
&\qquad r_i \in \{0, 1\} \qquad\qquad i = 1, \ldots, n \\
&\qquad k \in \{0, 1\}
\end{aligned}
\tag{2}
$$

where n is the number of robots, w_i is a weight for each robot i, which is 1 if the robot is loaded, and 0 if the robot is unloaded, and x_i and y_i are the location coordinates of each robot i. The variables r_i are binary variables, one per robot, which are forced to 1 by the "big M", $M \gg 0$, if the robot lies on the wrong side of the line. a, b, and c are the coefficients of the line we are trying to fit, where we add the constraint $a + b = 1 - 2k$, with k a binary variable: $a + b$ can only be 1 or -1, to exclude rescaling of solutions, and especially to avoid the solution where $a = b = c$.

This ILP tries to fit a line such that the two classes are as much as possible split by the line, and the objective value of the ILP gives the number of robots that are found on the wrong side of the line. In the reported results, we divide

this number of misplaced robots by the total number of robots in the swarm, in order to have the relative fraction value.

The imbalance evaluates whether the two classes are of the same size. We calculate imbalance are the ratio between the number of robots in the smallest of the two classes and the total number of robots in the swarm. Therefore, *the optimal value for imbalance is 0.5, the worst possible is 0.* Imbalance is important together with the linear separability. In difficult situations, loads often keep on travelling, and do not choose a place to settle, which leads to imbalanced solutions (more unloaded robots than loaded ones). In such cases, linear separability may be good (because there are few loaded nodes, so it is easy to split the two classes linearly), without the actual solution being good. Therefore, good linear separability needs to go together with good balance before we can decide there is a good solution.

In a first series of tests, we vary the *number of robots* in the swarm, from 10 up to 60. We keep the deployment area fixed to 3×3 m^2, and place the robots uniformly randomly. The communication range of the robots is limited to 1 m. The results are shown in Figure 4. They show that the algorithm works quite well, and is robust with respect to the number of robots, although for the smallest swarms, results become a bit less good. This is mainly because the network connectivity becomes worse, and also because there are too few loads around to be able to speak of proper clustering. The imbalance is less good, because with lower numbers of robots, the relative error when each robot randomly (with 50% probability) picks a class is higher. An example of a solution found by the robots, and the matching evaluation by the ILP is shown in Figure 5: for 10 robots, the solution is not so good, while for 50 it is of very good quality.

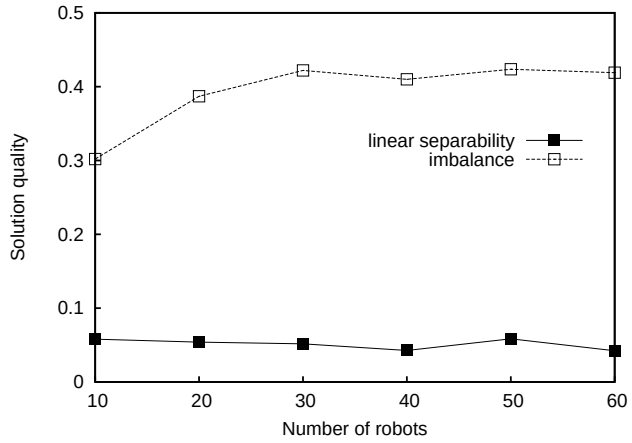

Fig. 4. Experiments with varying number of robots. The values for both linear separability and imbalance are reported.

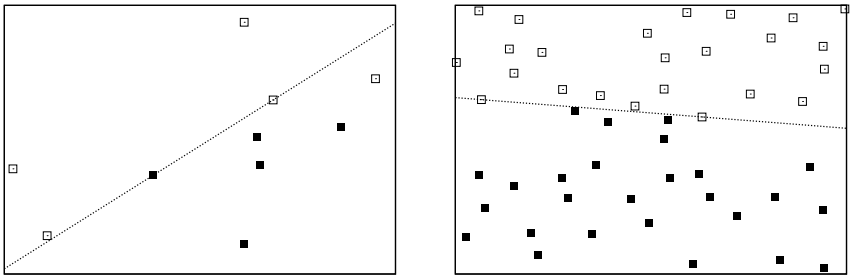

Fig. 5. Example scenarios with 10 (left) and with 50 (right) robots, after running the algorithm for 120 s. Black nodes indicate loaded robots, while white ones indicate unloaded robots. In each case, we draw the line that is found by the ILP in the evaluation of linear separability.

In a second series of tests, we vary the *communication range* of the robots, from 0.25 up to 4.5 m. The size of the arena is the same as before, and the number of robots is 50. The results are shown in Figure 6, the algorithm works badly at short communication ranges, due to the fact that the communication network gets disconnected. For medium and high communication ranges, the results are very good.

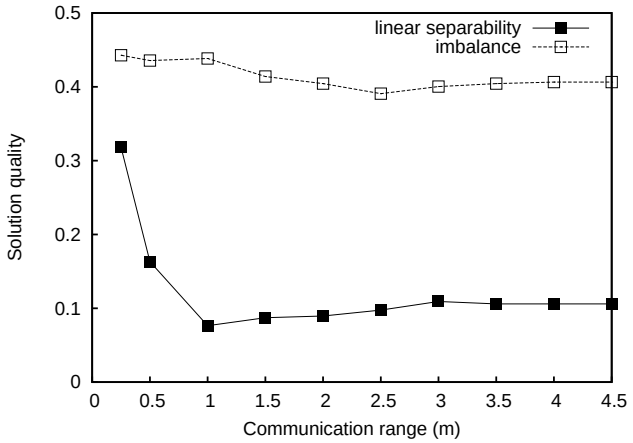

Fig. 6. Experiments with varying communication range maintaining fixed to 50 the number of robots in the swarm. The values for both linear separability and imbalance are reported.

The good performance at very high ranges (e.g., 4.5 m) surprised us a little, because in those situations, all robots are in range of each other, and the

algorithm has no way to identify local neighborhoods: all robots are each other's neighbor, and it is impossible to make local clustering. The results should therefore become very bad at very high ranges. Deeper investigation showed that the reason why behavior is good anyway was the fact that the communication channel used by the robots is based on infrared communication, which works only line of sight. As a consequence, robots that are further away from each other, have more probability of seeing their communication blocked, and so meaningful local neighborhoods are formed anyway. To test this, we also ran experiments in simulation where the communication was allowed even if the line of sight was blocked, with the results shown in Figure 7: here the performance indeed gets worse again for very high communication ranges.

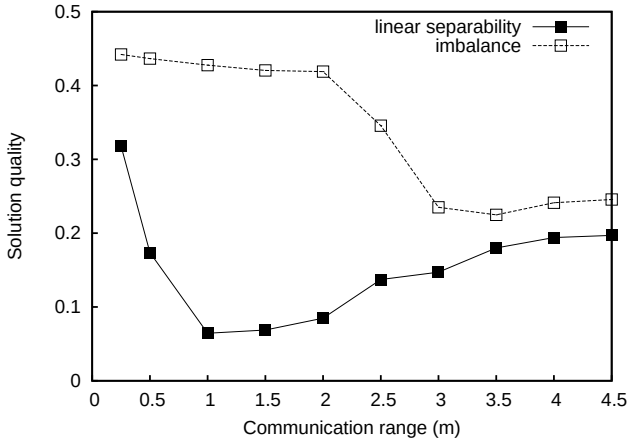

Fig. 7. Experiments with varying communication range. The line-of-sight constraint was removed: the communication between two robots is allowed if they are within their respective communication range. The values for both linear separability and imbalance are reported.

The graph in Figure 8 shows, for a typical example with 50 robots and range 1 m, how the solution quality, in terms of linear separability and imbalance evolves over time. Imbalance first goes down, as all loads start travelling. At the same time, linear separability also goes down. Then, loads start to settle down, and form clusters, where they stay more stably. As a consequence, imbalance goes up again (which means, the system gets more balanced), while the result from linear separation goes down (fewer misclassifications).

6 Implementation on real robots

We implemented the algorithm on real foot-bots, and carried out a few experiments with a small swarm of 15 foot-bots deployed in an open space of about

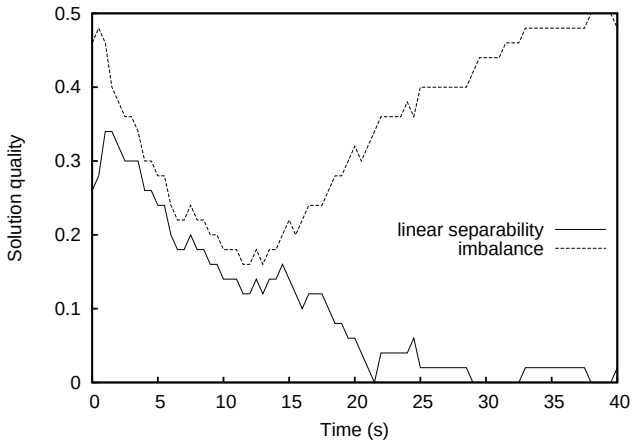

Fig. 8. Evolution over time of imbalance and linear separability for a test run with 50 robots and line-of-sight communication range of 1 m.

7×4 m^2. To allow meaningful testing in such a relatively small space, the line-of-sight communication range was set to 1.5 m. Moreover, to reduce the risk of packet losses, the maximum distance for a robot to be considered as a neighbor was set to 0.9 m. Some further adaptations were needed to make the system tested in simulation work on the real robots. First of all, we implemented an *acknowledgement* mechanism for loads, such that they can be transferred between robots over *lossy links*. A second change was to increase the default wait time from 40 to 120 seconds. Due to the fact that loads travel over lossy links, they go a bit slower, and hence we need to increase the wait time to keep the algorithm stable (otherwise, all loads would be travelling all the time). We also added an initial phase phase to identify the reliable neighbors, that is, to identify neighbors connected through reliable bidirectional links.

The acknowledgment mechanism works as follows. Each load gets an identifier from the sending robot (in the form of counter). Upon reception, the receiving robot sends an acknowledgment back to the sender, with the packet id. As long as the sender does not receive an acknowledgment with this id, he does not send more loads to this same neighbor. If it has not received an acknowledgment after a fixed number of time steps (set to 3), the robot resends this same load to the same neighbor. Each node also keeps for each neighbor the last received load id, to be able to check whether a load is new, or a duplicate (due to a lost acknowledgment message). This solution avoids that loads get lost in the network, or duplicated. However, the mechanism would require some further adaptation to work with unidirectional links, or in case of changes in the network connectivity (e.g., due to mobility). The structure of an acknowledgment message is packed in a 4-bytes packet, and contains:

– 1 byte for the message type,

– 1 bit containing u_i,
– 1 bit indicating whether the message contains a load or rather an acknowledgment,
– 1 byte encoding v_i,
– 3 bits indicating the number of acknowledgments included in the message,
– for each acknowledgment in the message: 1 byte indicating the target robot (sender of the original load) and 1 byte indicating the load id.

Figure 9, shows an example of the experimental setting and the evolution of the spreading of the load

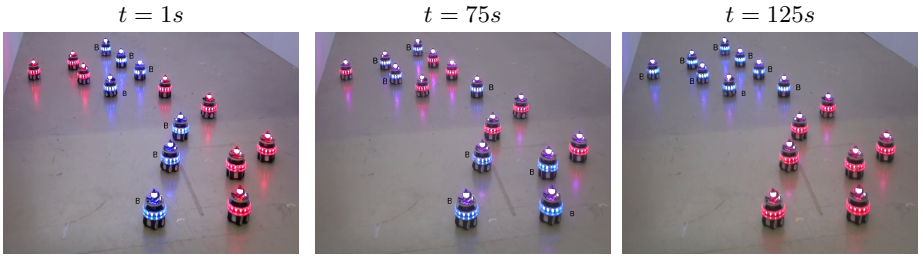

Fig. 9. Example of the evolution over time of the load/class distribution using a swarm of 15 real foot-bots. The line-of-sight communication range is set to 1.5 m. Starting from the random load configuration at time $t = 1$ (left sub-figure), the swarm reaches an optimal load distribution after 120 s (right sub-figure). Robots of class C_0 show blue LEDs, robots of class C_1 show red LEDs (to make the figure readable in greyscale a label 'B' is positioned beside each foot-bot in class C_0 to highlight its class membership, robots in class C_1 have no label). The obtained load distribution is optimal both in terms of linear separability and imbalance.

We ran a limited set of 10 experiments, testing different initial configurations of the swarm in terms of placement of the robots and using different random seeds for the initial definition of the load on each robot. We do not report here quantitative data since a larger set of experiments would be needed to get statistical significance, which is left as future work. However, by visual investigation, we could observe that in the large majority of the cases, the algorithm was able to determine an optimal, or near optimal spreading of the load, with the swarm reaching the wished spatial segregation in two classes. The time needed for convergence ranged from about 40 to 125 seconds. This relatively long time was mainly due to the large packet losses generated by the IrRB communication system, that determined the generation of a large number of acknowledgment packets and the multiple retransmissions of load messages.

A few sample videos, both for real robots and simulation can be accessed online at the following address:
`http://www.idsia.ch/~gianni/SwarmRobotics/GeometricSplitting.html`

7 Conclusions and future work

In this paper we presented a novel algorithm to let the robots in a swarm to reach spatial awareness for the purpose of clustering in two spatially segregated classes and being aware of the relative position in the cluster. The approach we proposed is fully distributed and only relies on local broadcast communications among the robots. The algorithm does not make use of any global positioning system and does not involve the use of mobility, for sake of speed of execution and saving of on-board energy. In a number of simulation tests we studied the performance of the algorithm considering linear separability and class imbalance as complementary metrics. The algorithm has shown good scalability versus the number of robots and robustness versus different communication ranges. We also provided a validation based on a small set of experiments with real robots.

As future work, we plan to make more extensive tests with real robots and modify the protocol to be more efficient in terms of robustness to packet losses and to increase convergence speed, which will be needed to be deal effectively with robot mobility. We plan to use the system in an application context in which spatial awareness and class aggregation will be functional to let the swarm cooperatively solve tasks such as the coordinated parallel exploration of multiple regions.

8 Acknowledgments

This research was partially supported by the Swiss National Science Foundation (SNSF) through the National Centre of Competence in Research (NCCR) Robotics.

References

1. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30:2826–2841, 2007.
2. R. Battiti and A. Bertossi. Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48(4), April 1999.
3. M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4187–4193, 2010.
4. J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 3(34):313–356, 2002.
5. M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. A. Di Caro, F. Ducatelle, E. Ferrante, A. Föster, J. M. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. M. de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling,

A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. Technical Report 14-11, IRIDIA, Brussels, Belgium, July 2011.

6. Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In *Proceedings of the 2nd ACM International Workshop on Principles of Mobile Computing (POMC)*, 2002.
7. P. Forero, A. Cano, and G. Giannakis. Convergence analysis of consensus-based distributed clustering. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1890–1893, 2010.
8. M. Franceschelli, A. Giua, and C. Seatzu. A gossip-based algorithm for discrete consensus over heterogeneous networks. *IEEE Transactions on Automatic Control*, 55(5):1244–1249, 2010.
9. F. Garcia, J. Solano, and I. Stojmenovic. Connectivity based k-hop clustering in wireless networks. *Telecommunication Systems*, 22(1):205–220, 2003.
10. S. Garnier, C. Jost, R. Jeanson, J. Gautrais, M. Asadpour, G. Caprari, and G. Theraulaz. Collective decision-making by a group of cockroach-like robots. In *Proceedings of IEEE Swarm Intelligence Symposium (SIS)*, pages 233–240, 2005.
11. A. Hayes, A. Martinoli, and R. Goodman. Swarm robotic odor localization: Off-line optimization and validation with real robots. *Robotica*, 21(4):427–441, 2003.
12. O. Holland and C. Melhuish. Stigmergy, self-organisation, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.
13. E. S. A. Jamalipour. Stable clustering and communications in pseudolinear highly mobile ad hoc networks. *IEEE Transactions on Vehicular Technology*, 57(6):3769–3777, November 2008.
14. V. Kawadia and P. Kumar. Power control and clustering in ad hoc networks. In *Proceedings of 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 459–469, 2003.
15. S. Kazadi, M. Chung, B. Lee, and R. Cho. On the dynamics of puck clustering systems. *Robotics and Autonomous Systems*, 46(1):1–27, 2004.
16. S. Kazadi, J. Wigglesworth, A. Grosz, A. Lim, and D. Vitullo. Swarm-mediated cluster-based construction. *Complex Systems*, 15(2):157–181, 2004.
17. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.
18. J. Kim, I. Hwang, Y.-H. Kim, and B.-R. Moon. Genetic approaches for graph partitioning: a survey. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 473–480, 2011.
19. M. Lehsaini, H. Guyennet, and M. Feham. CES: Cluster-based energy-efficient scheme for mobile wireless sensor networks. In *Wireless Sensor and Actor Networks*, volume II, pages 13–24. Springer, 2008.
20. A. Martinoli, A. Ijspeert, and F. Mondada. Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots. *Robotic and Autonomous Systems*, 29(1):51–63, 1999.
21. R. Olfati-Saber and J. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (CDC-ECC)*, pages 6698–6703, 2005.
22. C. Pinciroli, R. O'Grady, A. Christensen, and M. Dorigo. Self-organised recruitment in a heteregeneous swarm. In *Proceedings of the 14th International Conference on Advanced Robotics (ICAR)*, pages 1–8, 2009.

23. C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, T. Stirling, A. Gutiérrez, L. M. Gambardella, and M. Dorigo. ARGoS: a pluggable, multi-physics engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5027–5034, 2011.
24. J. Roberts, T. Stirling, J. Zufferey, and D. Floreano. 2.5D infrared range and bearing system for collective robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3659–3664, 2009.
25. T. Schmickl, R. Thenius, C. Moeslinger, G. Radspieler, S. Kernbach, M. Szymanski, and K. Crailsheim. Get in touch: cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, 18:133–155, 2009.
26. J. Werfel. Biologically realistic primitives for engineered morphogenesis. In *Proceedings of the 7th International Conference on Swarm intelligence (ANTS)*, pages 131–142, 2010.
27. M. Younis, M. Youssef, and K. Arisha. Energy-aware management in cluster-based sensor networks. *Computer Networks*, 43(5):649–668, 2003.
28. B. Zhou, A. Tiwari, K. Zhu, Y. Lu, M. Gerla, A. Ganguli, B.-H. Shen, and D. Krzysiak. Geo-based inter-domain routing (GIDR) protocol for MANETs. In *Proceedings of the IEEE Military Communications Conference (MILCOM'09)*, pages 1–7, 2009.