Self-Adaptive Pattern Formation with Battery-Powered Robot Swarms

Guannan Li
State Key Laboratory of Robotics
Shenyang Institute of Automation, CAS
Shenyang, China
University of Chinese Academy of Sciences
Beijing, China
Email: liguannan@sia.cn

Ivan Švogor
Department of Computer
and Software Engineering
Polytechnique Montréal
Montréal, QC, Canada
Email: ivan.svogor@polymtl.ca

Giovanni Beltrame
Department of Computer
and Software Engineering
Polytechnique Montréal
Montréal, QC, Canada
Email:giovanni.beltrame@polymtl.ca

***Abstract*—This paper presents a distributed, energy-aware algorithm for an autonomous deployment of battery–powered robots in a specified pattern. While each robot gradually discharges and leaves the formation to recharge, the algorithm presented in this paper assures that the formation pattern is preserved. This is achieved by defining the desired pattern as a point cloud where each point is occupied by a robot. The point cloud is transformed into a tree model that is shared among all robots. This model is used by each robot independently to govern its behavior, resulting in a self–adaptive network of robots which automatically generate paths for joining the formation and leaving it to recharge. Robots which leave the formation are replaced by neighbors to preserve the formation pattern. To demonstrate our algorithm we use a physics–based simulator and evaluate the persistence of the pattern formation formed by a robot swarm in an environment without global positioning, using only range and bearing.**

I. Introduction

Pattern formation with robot swarms is a popular research domain with various potential applications, e.g. providing infrastructural support for disaster rescue missions, distributed sensing, remote exploration etc. The application of robot swarms becomes particularly challenging in an environment without a global positioning, for instance in maritime research, ocean and space exploration. In such environments robots need to coordinate their actions without a central authority to oversee their activities or perform error corrections.

Hernandez-Martinez and Aranda-Bricaire presented a decentralized control algorithm in which robots are not aware of their global position and goals of other robots [6]. Using formation graphs and potential functions they deploy robots to achieve a formation. While this work is very similar to our general idea, the authors focus on a formal proof of global convergence to a desired formation using well defined formation graph properties of the Laplacian matrix, without energy constraints. With the same general assumptions, Fujinaga et.al. used a swarm of robots, using the cluster-weighted modeling algorithm and the Look-Compute-Move cycle [7]. Their paper presents the theoretical foundation and proofs of their approach, however with no considerations of physical limitations. They also emphasize that the time complexity of the formation problem as an open problem for the future work.

Mei et al. tackle the timing and energy consideration in this respect, however not with the goal of pattern formation, but rather to achieve coverage [8]. Their solution is the Space Partition Area Coverage Algorithm. This algorithm uses three main time related constraints: unloading time, dispersing time, and overlapping area. The authors present a comprehensive model of power consumption and time analysis to minimize the traveling distance for each robot to to achieve the desired coverage, the consumed energy, minimizing the number of robots. Their simulations show that they can reduce the number of robots by 30% while achieving coverage.

Derenick et al. present another work targeting energy concerns [5]. Their goal is to create a pattern formation with mobile robots and recharging capability using Centroidal Voronoi-Tessellation based control laws. With this approach, they obtain a critical battery level for each robot which guarantees the convergence of their solution. Their solution uses up to 7 robots per charging station, with the assumption of battery swapping rather than recharging.

Taking inspiration from nature to solve the pattern formation issue is also a popular approach. Guo et.al. developed a self-organizing algorithm for swarm robot pattern formation based on principles of gene regulation and cellular interactions [9, 10]. Each robot is treated as a cell, and the interactions between robots and the environment is modeled through reaction-diffusion mechanisms. They present a theoretical proof of algorithm convergence, a Matlab simulation along with real-world experiments with e-Puck robots. Similarly, Xu et.al. take their inspiration from biology to create an algorithm based on Particle Swarm Optimization and Virtual Pheromone mechanism [11]. In this solution, robots communicate only with their neighbors and when they reach a non–occupied area within the pattern, they label it and take position. Authors focus on large numbers of robots to test efficiency and scalability of their algorithm, i.e. pattern creation, not preservation.

Considering the work briefly presented above, one can notice that most of the authors take one of two approaches; a) developing an algorithm for pattern creation and its convergence proof and b) studying pattern maintenance with respect to physical constraints. In this paper we tackle both issues in

the context of a totally distributed system.

In this paper we assume such an environment and present a self–adaptive algorithm which deploys and maintains a swarm of robots in specified formation pattern. Compared to previous work, the challenge on which we focus in this paper is a limited energy source of the robots in a swarm and a task to deploy and keep a specified formation pattern. Furthermore the goal is to achieve this without the notion of global positioning, i.e. by using only range and bearing information obtained through communication with other robots. This approach is highly applicable in underwater robotics since it does not require access to expensive sensors like a Doppler velocity logger or an Inertial navigation system, and in space robotics where the global positioning system (GPS) is unavailable. Range and bearing information can be transferred through low bandwidth cheap light, radio or acoustic communication. Furthermore in this paper we also consider a limited communication range preventing all robots to communicate at once, which is why we resort to using situated communication [1].

We propose an algorithm which uses a point cloud input respresenting a desired pattern which a swarm needs to form. This point cloud is transformed into a tree model which is shared by all robots in the swarm and used to gain the knowledge about the pattern and the individual actions of each robot within the swarm necessary to form a desired pattern. These actions were defined using a finite state machine. When deployed, robots proceed to form a pattern as described in a share tree model through communication with their neighbors and reaching a consensus on their individual positions. The pattern formed by a swarm is a tree which defines a predecessor–successor relationship through which each robot has a replacement in the case when one of them needs to leave the formation to recharge.

The algorithm presented in this paper guarantees that robots are continuously recharged while keeping the pattern stable using only situated communication and local positioning.

The rest of the paper is organized as follows. In section II, we define the theoretical framework of our solution. We introduce the algorithm for generating a tree model in section III, and the behavior law as a finite state machine is introduced in section IV. Simulation results are shown in section V. Finally section VII concludes the paper.

II. Using a Robot Swarm for Pattern Formation

The swarm ecosystem we address in this paper consists of n robots and a charging station and a task to provide network coverage over a given area. To achieve this, robots need to form a pattern. The desired pattern is given in the form of a point cloud which is transformed into a tree model T, used by robots to calculate their local positions with respect to other robots. In this section, we explain how robots communicate and what information can be exchanged.

A. Behavior model

Given a pair of robots p and s, let p be the predecessor, and let s be its successor in the tree T. Assuming that p has already arrived to the target position while s has not, we use two local polar coordinates to navigate robot s to its target position. First local coordinate P^p with the center and reference heading of robot p and the second local coordinate P^g with the center and reference heading of robot s. Through situated communication, robot p gets the position of the robot s in its own frame of reference as $P_s^p(\alpha_s^p, d_s^p)$, where α_s^p is the bearing and d_s^p is the range. Similarly, robot s obtains the position of p in its own frame of reference as $P_p^s(\alpha_p^s, d_p^s)$. For s to reach its target position, we assume that there is also a global reference direction and a known angle α_{pg}^e which represents the error between the global reference direction and the local reference direction of the robot p. p sends P_s^p and α_{pg}^e to s. Using the tree T, s gets the information about the required distance and bearing with respect to p's global reference direction as $P_d^g(\alpha_d^g, d_d^g)$. Using this, it can navigate to its target position (known from the tree T).
Next, we present a set of behavior rules used by robots to reach their target positions.

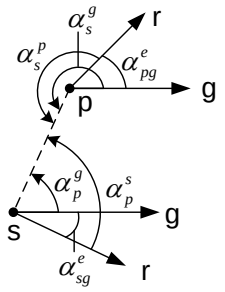

Fig. 1. Coordinate of a pair of predecessor and successor.

To explain how these rules are obtained, consider Fig. 1 which shows points s and p as current positions of two robots. $\vec{sg}$ and $\vec{pg}$ are the global reference directions which point in the same direction, $\vec{sg} = \vec{pg}$. Also, vectors $\vec{sr}$ and $\vec{pr}$ represent local reference directions of robots in consideration. Using these, we get:

$$|\alpha_s^g - \alpha_p^g| = \pi$$
$$\alpha_s^p = \alpha_s^g + \alpha_{pg}^e$$
$$\alpha_p^s = \alpha_p^g + \alpha_{sg}^e$$

where α_{sg}^e is the error between the local coordinate of s and the global reference direction. Notice that both α_{pg}^e and α_{sg}^e can be either positive or negative.
Let $\alpha_s^g \in [0, 2\pi]$ and $\alpha_p^g \in [0, 2\pi]$, then:

$$\begin{cases} \alpha_s^g - \alpha_p^g = \pi & \alpha_s^g > \pi \\ \alpha_p^g - \alpha_s^g = \pi & \alpha_s^g < \pi \end{cases}$$

so consequently:

$$\alpha_{sg}^e = \begin{cases} \pi + \alpha_p^s + \alpha_{pg}^e - \alpha_s^p & \alpha_s^p - \alpha_{pg}^e > \pi \\ \alpha_p^s - \pi - \alpha_s^p + \alpha_{pg}^e & \alpha_s^p - \alpha_{pg}^e < \pi \end{cases}$$

Therefore with a global reference direction, the vector from robot s to the target position P_d^g is:

$$\vec{P}^g_{sd}(\alpha^g_{sd}, d^g_{sd}) = \vec{P}^g_{sp} + \vec{P}^g_{pd}$$

where $\vec{P}^g_{sp}$ and $\vec{P}^g_{pd}$ are vectors from s to p and from p to s, respectively, with a global reference direction.
Given that $\vec{P}^g_{pd} = P^g_d$, and $\vec{P}^g_{sp} = (\alpha^s_p - \alpha^e_{sg}, d^s_p)$, in the local frame of reference of robot s the vector to its target position is:

$$\boldsymbol{u} = \vec{P}^s_{sd} = (\alpha^g_{sd} + \alpha^e_{sg}, d^g_{ds}).$$

The vector $\boldsymbol{u}$ is used as a control input for the robots actuators. Since this approach uses a tree model, a successor can be a predecessor for another robot. Then the error angle α^e_{sg} becomes α^e_{pg} for the successor of s. To navigate all robots to their positions, we assume that the charging station is the tree root, i.e. a robot already in its position. So, the local coordinate of the tree root is a global reference direction. Then, the first error angle is $\alpha^e_{pg} = 0$, and the remaining ones (α^e_{sg}) are calculated for each robot successively.

B. Charging model

For a vertex v_i in tree T, let E_i be its energy. We define the operation *exchange* for a pair of neighboring vertices v_a and v_b, with v_a being the predecessor of v_b (and consequently v_b being the successor of v_a). If $E_a > E_b$, meaning that a predecessor has more energy than its successor, v_a and v_b invoke the *exchange* operation which swaps their positions.

With this, v_a becomes a successor while v_b becomes its predecessor, resulting with a state in which a predecessor has less energy than a successor. If v_0 is the charging station, it is only allowed to have one successor v_1 (i.e. only one robot can recharge at a time) which overlaps with its position meaning that it is getting recharged.

Theorem 1. *Continuous invocation of the operation* exchange *on T, where $|T| = n$, T will converge to $E_1 \leq E_i (i = 2, ..., n)$.*

Proof. Assume that vertex v_e has the lowest energy, namely: $E_e < E_i (i \in \{1, ..., n\} \setminus \{e\})$ and $E_e > 0$. Consider the path from v_0 to v_e and let the distance between v_0 to v_e be given as $d_{0,e}$. Applying *exchange* on each pair of neighbors in path $v_0 v_e$ is called a *round*. After each *round*, $d_{0,e}$ will decrease by 1, therefore by continuing the application of *exchange*, $d_{0,e}$ will decrease to the $\min d_{0,e} = 1$ because $E_e > E_0 = 0$. Finally, v_e will become v_1, i.e. $E_1 \leq E_i (i = 2, ..., n)$. □

From the theorem above, we have:

Corollary 1.1. *If T' is a subtree of T obtained by cutting leaves, a sufficient application of* exchange *operation will result in $E_1 \leq E_i (i = 2, ..., n)$.*

This implies that the position exchanging process between robots can start even before the pattern formation is complete.

Corollary 1.2. *By applying the operation* exchange *on a directed path in T, T will converge to the state where $E_m < E_n$ if $d_{0,m} < d_{0,n}$.*

From the previous corollary we know that a robot will be pushed towards the edge of the tree once fully charged, while robots with less energy will be attracted towards the root.

III. The tree model algorithm

In this section we will introduce the algorithm for generating the tree model T, using the point cloud G (Algorithm 1).

The point cloud G is an input to the algorithm and it is transformed into a directed graph T (which is a tree, the proof follows). Then, two empty lists are created L_u and L_l. L_u (line 1) is used to store points that have not been added into the graph T, we call these the *unLabeled* points, while L_l is used to store the points already added to the graph T, i.e. the *labeled* points. Whenever a point is added to T the counter I_{index} is incremented, starting from 0.

The algorithm starts by adding the charging station (point v_0) to T, and appending it to the list L_l (line 2).

Algorithm 1 Generate tree model from point cloud

Input: Point cloud G
Output: Tree model T recorded in table
1: unLabeled list $L_u = \phi$, Labeled list $L_l = \phi$, $I_{index} = 0$
2: $L_l.push(v_0)$, RECORD(v_0,I_{index}, -1), $I_{index} = I_{index} + 1$
3: Sort all points in G along distance to v_0 from the nearest to furthest, as $V\{v_1, v_2, ..., v_n\}$
4: $L_u \leftarrow V\{v_1, v_2, ..., v_n\}$
5: **while** $L_u \neq \phi$ **do**
6: Candidate point:$v_c \leftarrow L_u.pop()$
7: Create set $P_c = \{v_{pc}$: no point exists in line segment $v_{pc}v_c, v_{pc} \in L_l\}$
8: Pick the point nearest to v_c in P_c as the predecessor of v_c, call it v_p
9: $L_l.push(v_c)$, RECORD(v_c,I_{index},v_p)
10: $I_{index} = I_{index} + 1$
11: **end while**
12:
13: **function** RECORD(v_{self},L_{label},$v_{predecessor}$)
14: **if** the label of $v_{predecessor} \neq -1$ **then**
15: Calculate the distance and bearing of v_{self} w.r.t $v_{predecessor}$, as d and α
16: Add a row to the table, as: $Lab = L_{label}$, $Pred =$ the label of $v_{predecessor}$, $Dis = d$, $Bearing = \alpha$
17: **else**
18: Add a row to the table, as: $Lab = L_{label}$, $Pred = -1$, $Dis = -1$, $Bearing = -1$
19: **end if**
20: **end function**
21:
22: **function** PUSH(v_{point})
23: Put v_{point} at the end of the sequence
24: **end function**
25: **function** POP()
26: Pick the first point out of the sequence
27: **end function**

On line three, all points in G are sorted (including a copy of v_0) with respect to distance from v_0, from nearest to furthest as $V\{v_1, v_2, ..., v_n\}$. In line four the points are added into the list L_u to represent candidate points waiting to be added to the graph T.
Finally, a loop (lines 5–11) iterates through points in the list L_u which represent candidate points (v_c) to enter the graph. It

also iterates through a set of points in the list of added points L_l which are its potential predecessors. If a point v_{pc} in L_l can be seen by v_c (no points exists in the line segment between v_c and v_{pc}), it becomes a potential predecessor. The one nearest to v_c becomes its predecessor and once a predecessor is found, v_c is moved from the list L_u to to the list L_l. For any v_c, a predecessor can be found, meaning that all points in G can be added into T:

Theorem 2. *The output of Algorithm 1 contains all points in G.*

Proof. While iterating through the tree T, v_c cannot be added to T only when $P_c = \phi$. Assuming that $P_c = \phi$, then for the current L_l, $\forall v_l \in L_l$, $\exists v_{l,1}, ..., v_{l,k} \in L_u (k \geq 1)$ lie between v_c and v_l. Since we always have $v_0 \in L_l$, then $\exists v_{0,1}, ..., v_{0,k} \in L_u (k \geq 1)$ lie between v_0 and v_c. Let $v_{0,1}, ..., v_{0,k}$ be a sequence sorted according to distance to v_0 from near to far, then sequence $v_{0,1}, ..., v_{0,k}, v_c$ is also sorted according to distance to v_0 from near to far. This means before testing v_0, we already test if $v_{0,1}, ..., v_{0,k}$ can be added to T. As no point belong to L_u exists between v_0 and $v_{0,1}$, point $v_{0,1}$ must be added to T. Similarly, before testing v_c, $v_{0,1}, ..., v_{0,k}$ have already been added to T, which contradicts to the assumption, $P_c \neq \phi$. This means every point in G can be added to T. □

Theorem 3. *The output of Algorithm 1 is a tree.*

Proof. When $L_l = v_0$, v_1 is placed into the graph T in accordance to the algorithm resulting in a graph G_2 which is clearly a tree at this point. Furthermore when $L_l = v_0, v_1, ..., v_{n-1}$, the graph G_n is still a tree. By adding a point v_n to G_n, letting its predecessor be the point v_p, we can get G_{n+1}. Since G_n is a tree, does not contain cycles. Adding a vertex and a edge to G_n cannot result in a cycle in G_{n+1}. As G_n is a tree, it is a connected graph and there exits a path between each pair of vertexes. So $\forall v_i (i = 0, 1, ..., n-1, i \neq p)$, there is a path between v_i and v_p. So there is a path between v_n and v_i, meaning that G_{n+1} is also a connected graph. A connected graph without cycles is a tree, so finally T is a tree. □

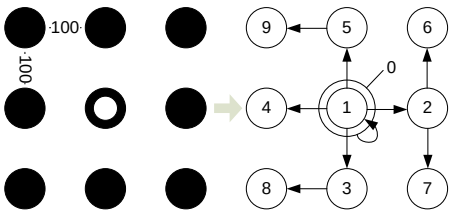

Fig. 2. In put the point cloud in the left into the algorithm, getting the output tree in the right.

Fig. 2 shows a point cloud in the left in which the white ring represents a base station. The generated tree T is shown on the right. The number on the left side of the image represent distances between the points, while the numbers on the right side indicates the Label of each vertex. Arrows are pointing from a successor to its predecessor. Notice that the vertices labeled 0 and 1 are overlapped representing a robot in the charging station, which is also the part of the tree.

TABLE I
THE TREE IN FIG. 2 RECORDED AS A TABLE

Lab	Pred	Dis/(cm)	Bearing/(°)
0	-1	-1	-1
1	0	0	0
2	1	100	0
3	1	100	270
4	1	100	180
5	1	100	90
6	2	100	90
7	2	100	270
8	3	100	180
9	5	100	180

For the tree T to be usable by robots it is represented in the form of a table shown in Table I. It contains four items; 1) the label of a vertex Lab, 2) the predecessor $Pred$ of vertex Lab, 3) the distance Dis and 4) the bearing $Bearing$ of vertex Lab with respect to its predecessor. All vertexes in the table share the same reference direction for the bearing. The first vertex has no predecessor which is denoted with -1. Once a point is added into L_l, the function *Record()* adds the corresponding record into the table (lines 19–20).

IV. BEHAVIOR

A. Finite state machine

The behavior rules of robots is represented using the finite state machine (FSM) shown in Fig. 3. The finite state machine consists of seven states, and a charged robot is always in one of the following: 1) **Free**, 2) **Asking**, 3) **Joining**, 4) **Joined**, 5) **Upasking**, 6) **Goup** and finally 7) **Godown**. Its transition conditions are given in Table II.

TABLE II
STATE TRANSITION CONDITIONS

Index	Condition
(1)	Get a proper label
(2)	Label application is rejected
(3)	Label application is accepted
(4)	Communication with predecessor is lost for a while
(5)	Arrived at the target position
(6)	Application to start the exchange process is approved
(7)	A successor is taking the place of its predecessor
(8)	Robot has less energy than its predecessor or the lower threshold
(9)	The exchange process is denied
(10)	The exchange process is approved
(11)	Arrived at the target position

The charging station is treated as a special robot which also runs the same FSM, however it always stays in the state **Joined**, and it is assigned to the label 0. Initially all robots are in the state **Free** and they need to obtain the label of the

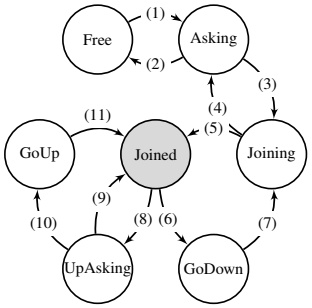

Fig. 3. The behavior law represented as a finite state machine.

vertex in the tree T through the interaction with neighbors during the pattern formation process.

The FSM has two paths for each robot to engage with. The first path is the deployment path, while the second one is the charging path. By engaging with the deployment path, a robot in the state **Free** obtains a label and becomes the part of the pattern. Likewise, by engaging with the charging path, robots exchange positions in order to reach the charging station.

B. Deployment path

The deployment path consists of the following states: **Free**, **Asking**, **Joining**, **Joined**.

Free: a robot in state **Free** does not have a label, meaning it still isn't the part of the pattern. A **Free** robot broadcasts its state, and moves around robots which are already within the pattern, i.e. those in state **Joining** or **Joined**. To achieve that behavior, two forces are applied on the robot. The first one keeps the robot at a certain distance to robots already in the pattern, while the second one (vertical to it) will move the robot around the formed pattern.

To translate these forces in to a navigation rule, consider a **Free** robot i and a set J of k robots which are in the state **Joining** or **Joined**. In the frame of reference of robot i the position of a robot $j \in J$ is (α_j^i, d_j^i). Following this, a navigation behavior can be obtained as:

$$\boldsymbol{u_i} = f\left(\frac{1}{k}\sum_{j=1}^{k}(\alpha_j^i, d_j^i - d_o) + \frac{1}{k}\sum(\alpha_j^i + \frac{\pi}{2}, d_m)\right)$$

where d_o and d_m are the strength of the force field. The function function $f(\cdot)$ maps the resulting force to a proper control input of the robot.

If a robot in the state **Free** finds a label which is not occupied by another robot, and it can see a predecessor of this label already in the state **Joined**, it will transit to state **Asking**.

Asking: a robot in this state still does not have a label, however it broadcasts a message to apply for one. When a broadcast message is confirmed, it transits to the state **Joining**. If the robot is rejected however, it transits back to the state **Free**. This can happen if multiple robots apply for the same label, or when a potential predecessor is no longer in the state **Joined** (i.e. it is engaged in the position exchange) The navigation rule of the robot in the **Asking** state is the same as in the **Free** state.

Joining: state indicates that robot has a label, i.e. it knows its target position with respect to its predecessor. A robot in this state uses a control rule as the one described in section II. To reach the target position, its predecessor repeatedly broadcasts *guidance* messages to each of its successors in the state **Joining**. A *guidance* message contains 1) the label of the **Joining** successor, 2) the relative position of the **Joining** successor and 3) the error between global reference direction and the local position of the predecessor, α_{pg}^e. Using this information, a robot in the state **Joining** calculates its control input $\boldsymbol{u}$. A robot in this state also broadcasts its state and label so that its neighbors are aware of its presence. If the communication with the predecessor is lost (timeout), a robot will transit back to state **Asking**.

Joined: a robot in the state **Joining** will transit to this state once it arrives to the target position. A robot which is in the pattern formation will hold still and perform any processing which might be necessary (depending on the task and the goal of the swarm). Also, in this state a robot is receiving messages from its neighbors, and when it receives a message from a robot in the **Asking** state it either confirms or rejects it, depending if it already has a successor. If it does not have a successor and multiple robots apply, it chooses one (the first in our case) and starts sending the *guidance* messages.

C. Charging path

The charging path is realized through the position exchange between pairs of robots, a predecessor and a successor, until finally the one with the least amount of energy reaches the charging station. Its successor takes the place of its predecessor, through a *Go Up* process while a predecessor takes the palace of its successor through a *Go Down* process. The FSM defines the charging path with two sub–paths shown in Fig. 3.

The *Go Up* path consists of the following states:

Joined: a robot in the **Joined** state listens for the messages from its predecessor and compares its energy level to its own. If it has less energy than its predecessor or the lower threshold, robot transits to state **UpAsking**.

UpAsking: a robot in state **UpAsking** proceeds sending messages to its predecessor to trigger the *exchange* process. If a reply is received from the predecessor, confirming the *exchange* process, it transits to state **GoUp**. However, if it is refused, it transits back to state **Joined**. This can happen when a predecessor is guiding a robot in the state **Joining** or **GoUp** towards the target position. Furthermore, if a robot in the **UpAsking** state does not receive a response, it will also transit to the **Joined** state. This happens when multiple successors try to trigger the exchange process, and the predecessor confirmed the application from another successor.

GoUp: when a robot transits to this state from state **UpAsking**, it changes its label to the label of its predecessor and then, its current predecessor (the predecessor of its original predecessor, i.e. grandfather) is guiding it to its target positions using *guidance* messages. The navigation rule depends on the message from a current predecessor. If a robot can see its current predecessor, the navigation rule is the same as for the

state **Joining** and if not a robot will be guided by the original predecessor, who will move away so that the **GoUp**robot see its new predecessor.

Joined: when finally the robot in the state **GoUp** arrives at the target position, it will transit to the **Joined** state. And if the target position is the charging station, it starts to recharge. Along the *Go Down* path, a predecessor will encounter:

Joined: when a robot in the **Joined** state receives a message from a successor in the **UpAsking** to exchange positions, it needs to decide whether it should confirm or reject it. If a robot in the state **Joined** has a at least one successor in the state **Joining**, **GoUp** or **GoDown** it should reject the request to start the exchange process. Otherwise, it should confirm the position exchange request, from its successor with the energy level. If this happens it transits to the state **GoDown**. The robot in charging will reject all requests if it has less energy than the upper threshold.

GoDown: a robot in the **GoDown** state starts sending messages to apply for a label of its successor. When it is accepted, a robot will transit to the state **Joining** and obtain a label. The navigation rule of the robot in the **GoDown** state is such that it moves a bit to make space for its successor in the the **GoUp** state. After its successor takes its position, it will start moving towards the old position of its successor.

Joining: When the application of a **GoDown** robot is confirmed, it will be guided to the target position in the same manor as in the **Joining** state. It will also transit to the state **Joined** once arrives at destination.

Joined: the *Go Down* path ends at a **Joined** state.

The exchange process repeats and finally the robot with the lowest energy will move to the charging station, and robot has just been charged will be pushed to the edge of the tree T.

V. Simulation

This section presents a setup of a simulation used to verify the theoretical framework presented previously, along with simulation results and a brief discussion.

A. Simulation environment and assumptions

As a simulation environment we used ARGoS (Fig. 8), a multi–physics robot simulator along with Buzz, which is a domain specific language designed in particularly for programming swarm behavior[2, 3]. The robot used in the simulations is a customized version of FootBot which includes additional LEDs and a battery model. A battery model allows us to simulate its charging and discharging [4].

To validate the presented algorithm and evaluate the persistence of formation pattern we performed multiple thorough and extensive simulations. We consider a scenario in which a swarm of robots has a goal to form a pattern over a certain area, and once the pattern is formed, each robot acts as a base station to provide network coverage. We simulate this by using different discharge currents.

The formation is considered to be stable if none of the robots fully discharges. However, the system has a physical constraint related to its energy input. Looking at it as a black box, for the swarm to be stable, its energy input must be more and equal to its energy consumption. This means that given the charging time t_c using the charging current I_c, and the discharging time t_d using the discharging current I_d the condition $I_d \cdot t_d < I_c \cdot t_c$ must be satisfied. This implies that the charging time should always be shorter than discharging time, and the degree of freedom which can control this is the charging current, i.e. with its increase the total charging time can be reduced. Consequently, introducing n robots into a swarm, the charging time needs to decrease n times, i.e. the charging current must be such that $n \cdot I_d + c$, where c assures the reserve power (necessary to form the pattern).

This constraint is in direct relation to the properties of the battery, its capacity and charging C rating [4]. For simulation purpose, we assume that the battery supports fast charging. However, it can also be assumed that at the charging station, the battery is physically replaced [5].

B. Simulation setup

The simulation starts with all robots in the idle state, after which the pattern formation algorithm starts the deployment. Depending on robots current state, its battery is discharged with three different currents shown in Table III. For example, if a robot is moving towards its target position its battery is discharged with 1 A, since when it is moving it consumes 0.8 A, and when its idle (on-board computers and sensors) it consumes 0.2 A. In the state **Joined** the discharge current is such that it simulates heavy processing by on-board computers, i.e. providing network coverage in accordance to the considered scenario. The battery model is such that it represents a single celled battery with 4.2 V, and a reduced capacity of 200 mAh, to keep the duration of simulations reasonable.

TABLE III
Battery model properties and average current consumption (ACC)

ACC Idle (supporting on-board electronics)	ACC Driving	ACC state **Joined**	Battery capacity
0.2 A	0.8 A	1.2 A	200 mAh

We have learned experimentally that 8000 simulation steps are sufficient to verify or demonstrate the stability of the system. With simulation step set to 0.1 S, the run–time of each simulation translates to 8 minutes in the real world.

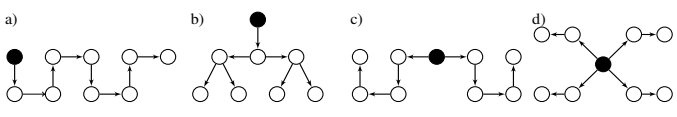

Fig. 4. Pattern formations used in simulations. Black node represents a charging station, and they are named *a, b, c, d*; from left to right respectively.

The simulation was performed using four different pattern formations shown in Fig. 4 which were selected with following concerns a) depth of the tree (which translates to number of position exchanges for a robot to reach the power station

and to the maximal number of possible position exchanges at any given moment) and b) the position and number of direct successors from the charging station (which translates to the number of robots competing to recharge). In this paper, only the pattern formation *c* is presented, while the remaining ones, along with all collected data from simulations are available on the open science data repository FigShare[1].

C. Results and discussion

Fig. 5 shows the state of charge (SoC) for each robot while simulating the pattern formation *c*. All robots start fully charged and gradually discharge. Each time when a robot has a higher SoC than its successor, they exchange positions. When reaching a charging station, robots SoC starts gradually increasing until the next robot (its first successor) takes its place in accordance to Algorithm 1. Fig. 5 shows that the system is stable with (at most) 9 robots, while the SoC never drops under 50%, meaning that there is some power reserve which allows for a decrease of charging current.

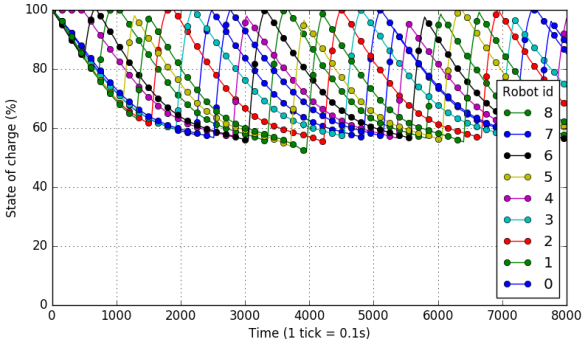

Fig. 5. State of charge for each of the 9 robots in 8000 simulation steps.

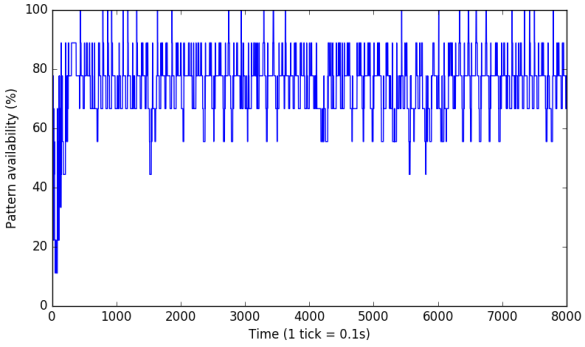

Fig. 6. Availability of the desired pattern formation

Simulations have shown that with introduction of new robots, the pattern formation and the simulated network coverage stability will not increase. This is due the fact that the proposed algorithm insists on continuous and immediate exchange of positions between a predecessor and a successor when there is a SoC difference. As Fig. 6 shows this doesn't completely impair the formation. In fact, 60–80% of robots are at any given time at their exact position as the desired pattern requires, while the rest of them are engaged in the position exchange process. In absolute terms this means that 2 to 3 robots are exchanging positions. As for the stability of the simulated network coverage, figure Fig. 7 shows that on average 80% robots are keeping the established network alive, meaning that on average only 1 or 2 are not in the pattern or at a nearby position. This does not mean that the coverage is completely unavailable since the a predecessor only moves slightly from its position in order to make space for its successor. Until this happens, a predecessor is acting as a base station (i.e. simulating data processing).

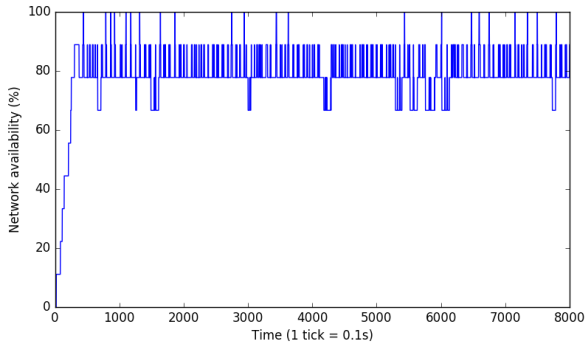

Fig. 7. Availability of robots within the pattern. Network is unavailable if any of the positions within the pattern are not occupied by a robot.

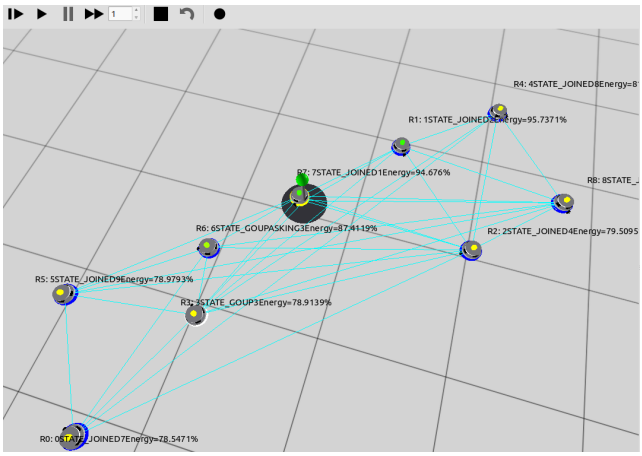

Fig. 8. ARGoS capture showing formation pattern *c*. R7 is charging, while R3 with less power is exchanging the position with R6 which has more power.

Fig. 8 is a screen capture from ARGoS which shows the robots in the pattern formation *c*. At the particular moment of screen capture, robots R6 and R3 are exchanging positions. R3 is on its way to the position of R6, while R6 is performing its task (acting as a base station).

Since the charging station is positioned in the middle the depth of pattern formation *c* is 4, meaning that robots at leaf positions need to make four position exchanges in order to reach the charging station. And since the charging station has two immediate branches, two robots are competing to get charged at the same time. Given its depth and number of

[1]Remaining experiments are available at https://doi.org/10.6084/m9.figshare.4776637

direct branches from the root, pattern formation *c* has a good representation of properties of all other patterns, and this is why it is chosen to be presented in this paper.

Performing simulations with the remaining patterns has shown that pattern *a* is the hardest one to keep stable, since it has the highest depth. For a robot to reach the charging station it requires 9 position exchanges, and given the parameters in Table III it was stable with only 6 robots. Pattern *b* was stable with 7 robots, and interestingly it had the highest percentage of robots in the formation in any given moment. This is since the exchange process between the charging station and its successor momentarily blocks all other exchanges. Finally, the pattern *d*, with depth 2 and its topology allowed for at most 4 position exchanges while being stable with 9 robots.

As a concluding remark, notice that providing network coverage and keeping the pattern formation stable are conflicting requirements. In one hand the full network coverage is not available when robots are exchanging positions and on the other hand robots need to exchange positions in order to recharge. Also, from a physical standpoint the general condition for system stability is $I_d \cdot t_d < I_c \cdot t_c$, while the pattern stability also depends on tree model topology.

VI. Conclusion

This paper presents an energy–aware robot deployment and pattern formation algorithm which uses only local positioning to achieve it. In addition, the presented algorithm addresses both energy concerns and sensory limitations which can be expected in space, underwater and remote environments. To overcome this, a point cloud which represents a desired pattern is transformed into a tree model which is shared by all robots in a swarm. Each robot takes a place within the tree model by communicating through situated communication only with their neighbors with messages containing only range, bearing and battery level information. Employing a set of behavior rules in the form of a finite state machine, each robot coordinates its own actions necessary to create the pattern and keep it. If any of the robots in the formation has an energy level lower that its predecessor (known from the shared tree model) they exchange places. This continues until finally the robot with the least energy gets to the root, i.e. the charging station. To validate our algorithm, we provide a formal proof that it converges, along with a series of simulation which demonstrate it and show that the pattern formations are stable and none of the robots are fully discharged (following a set of conditions).

Simulations have also shown that there is a great potential for future work, primarily considering a non–uniform discharge, deeper investigation into the tree model topology and also considering a healing scheme for robots which get fully discharged. We are also working on expanding the tree model into a forest model so that multiple charging stations will be involved in the system, expanding the scale of the pattern. Finally, our plan is to test it on a real–world system with a heterogeneous swarm.

References

[1] K. Støy, "Using situated communication in distributed autonomous mobile robots," *Proceedings of the 7th Scandinavian Conference on Artificial Intelligence*, pp. 44–52, 2001.

[2] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[3] C. Pinciroli and G. Beltrame, "Buzz: An extensible programming language for heterogeneous swarm robotics," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2016, pp. 3794–3800.

[4] M. Coleman, C. K. Lee, C. Zhu, and W. G. Hurley, "State-of-charge determination from emf voltage estimation: Using impedance, terminal voltage, and current for lead-acid and lithium-ion batteries," *IEEE Transactions on industrial electronics*, vol. 54, no. 5, pp. 2550–2557, 2007.

[5] J. Derenick, N. Michael, and V. Kumar, "Energy-aware coverage control with docking for robot teams," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3667–3672, 2011.

[6] E. Hernandez-Martinez and E. Aranda-Bricaire, "Decentralized formation control of multi-agent robots systems based on formation graphs," *Studies in Informatics and Control*, vol. 21, no. 1, pp. 7–16, 2012.

[7] N. Fujinaga, Y. Yamauchi, S. Kijima, and M. Yamashita, "Asynchronous pattern formation by anonymous oblivious mobile robots," *Lecture Notes in Computer Science*, vol. 7611 LNCS, pp. 312–325, 2012.

[8] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "Deployment of mobile robots with energy and timing constraints," *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 507–522, 2006.

[9] H. Guo, Y. Meng, and Y. Jin, "Swarm robot pattern formation using a morphogenetic multi-cellular based self-organizing algorithm," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3205–3210.

[10] Y. Jin, S. Member, H. Guo, and Y. Meng, "A hierarchical gene regulatory network for adaptive multirobot pattern formation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 805–816, 2012.

[11] H. X. H. Xu, H. G. H. Guan, A. L. A. Liang, and X. Y. X. Yan, "A Multi-robot Pattern Formation Algorithm Based on Distributed Swarm Intelligence," *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*, pp. 71–75, 2010.