Optimization of the Logical Topology for Mobile MEMS Networks

Hicham Lakhlef[a,*], Hakim Mabed[a], Julien Bourgeois[a]

[a] *UFC/FEMTO-ST, UMR CNRS 6174, 1 cours Leprince-Ringuet, 25201 Montbeliard, France*

Abstract

In this paper, we propose an improvement of the logical topology by using self-reconfiguration protocol in MEMS microrobot networks. Recently, solutions have been given for this problem for a chain of microrobots as a starting physical topology; the advantage of these solutions is that they are distributed protocols without map of the target shape which makes them efficient and scalable. This paper shows how to generalize the solution for any starting physical topology. We propose an efficient map-less self-reconfiguration protocol where nodes can perform the algorithm regardless the place where they are deployed, because the algorithm is independent of the map of the target shape. Furthermore, our solution tries to reach the target shape with a minimum amount of movement. The protocol consists of two main algorithms, *initiator election* and *shape shifting*. The initiator election aims to choose the best initiator that initializes the algorithm *shape shifting*, the criterion of choice is the effect on the amount of movement on the other nodes. The shape shifting algorithm aims to convert the initial physical topology with an incremental process using collaboration, coordination and help between nodes. This protocol is implemented in DPRSim, the Dynamic Physical Rendering Simulator.

Keywords: MEMS Microrobots; DiMEMS; Mobility; Self-reconfiguration; Self-organization; Physical Topology; Logical Topology; Communication; Energy

1. Introduction

Micro electro mechanical system (MEMS, for short) is a technology that enables the batch fabrication of miniature mechanical structures, devices, and systems that can sense and act [1, 2]. It is projected that these small devices, referred to as MEMS nodes, will be mass-produced, making their production cost almost negligible due to the microtechnology manufacturing. Their applications will require a massive deployment of nodes, thousands or even millions [3, 4] which will give birth to the Distributed Intelligent MEMS [5].

The size of MEMS microrobots can vary from well below one micron to several millimeters. Likewise, the types of MEMS devices can vary from relatively simple structures having no mobile elements, to extremely complex electromechanical systems with multiple mobile elements under the control of integrated microelectronics components. Due to their small size and the batch-fabrication process, microrobots are potentially very cheap, particularly through their use in many areas in our day to day life [6].

The Smart Surface project [5] is an example of MEMS system without mobile components. In this project, MEMS nodes form a planar surface and collaborate in order to recognize and to orient set of objects placed on it. In the Smart Blocks project [7], a modular blocks composed of several MEMS are able to move in order to transport some objects. The Claytronics Project [8, 9, 10, 11] studied in this paper, presents an example of mobile microrobots called *Catoms* in which direct communications are only possible between physical neighbors. In all these projects, there are at least some elements which have some sort of mechanical functionality whether or not these elements are mobile.

One of the major technological challenges for microrobot developing is the way to achieve a precise movement to reach a destination position using a very limited power supply. Many different solutions have been studied. Within the *Claytronics* project, microrobots help each other to know the positions reached which introduce the idea of a collaborative way of moving. But, even if the requested power for moving has been lowered, it still costs a lot regarding the communication and computation requirements [12]. Optimizing the number of movements of microrobots is therefore crucial in order to save energy.

At the present time, swarm MEMS nodes is gaining increasing attention since large-scale swarms can perform various missions and tasks, in a wide range of applications including odor localization, firefighting, medical service, surveillance and security, and search and rescue [13]. In order to help the microrobots swarm in achieving their mission, it is suitable to reorganize the geometric shape of microrobots according to their task as well as the en-

*Corresponding author

Email addresses: hlakhlef@femto-st.fr(H.Lakhlef), hmabed@femto-st.fr (H.Mabed), julien.bourgeois@femto-st.fr(J.Bourgeois)

vironment. Self-reconfiguration problem and construction of optimal logical topology (in term of message exchanging complexity) starting from a random physical topology is a very challenging problem. Most of the researches in the field of swarm configurations have not yet considered the energy and memory (needed data to map the desired shape) optimization aspects [14]. Our research therefore aims to develop self-configurable microrobot swarms that can be deployed in a constrained environment. Specifically, we have made the following assumptions to improve the literature works:

- no map of the target shape,
- direct communications are restricted to the immediate vicinity of the node (physical neighbor),
- the same algorithm is run on every node,
- and the sensing range is limited.

Based on these assumptions, we address here the problem of microrobots swarm reorganization. The adaptive self-configuration enables microrobots swarms to strive toward achieving their mission without map managing (predefined final positions) of the target shape.

Self-reconfiguration of microrobots is a useful distributed algorithm with many applications. Optimizing the energy cost of self-reconfiguration algorithms will, therefore, have a direct impact on the energy efficiency of the swarm.

In the literature, self-reconfiguration can be seen from two points of view. On the one hand, it is defined as a protocol, centralized or distributed, which reorganizes a set of nodes to reach an optimal logical topology [15]. In our works, the square shape was chosen since it represents an optimal physical topology for message exchanging. For example, a set of n connected microrobots organized as a chain presents a broadcast complexity of $O(n)$ in the worst case (required time to reach all the other nodes). When the n microrobots are reconfigured into a square shape, the worst case broadcasting complexity diminishes to $O(\sqrt{n})$. That is, the number of direct contacts between microrobots is minimal and secondly the average distance between two microrobots (in term of number of hops) is of $(n+1)/3$ where n is the number of microrobots. Therefore, improving the logical topology for communication will improve the propagation procedures and convergence in the network. On the other hand, and in second sense the self-reconfiguration is built from modules which are autonomously able to change the way they are connected, thus changing the overall shape of the network [10, 16].

The self-reconfiguration process is difficult to control, because it involves the distributed coordination of a large numbers of identical modules connected in time-varying ways. The range of exchanged information and the amount of displacements determine the communication and energy complexity of the distributed protocol. When the information exchange involves close neighbors (restricted number of hops), the complexity is moderate and the resulting distributed self-reconfiguration scales gracefully with network size if the protocol is without map of the target shape. An open issue is whether distributed self-reconfiguration would result in an optimal configuration with a moderate complexity in message, movements and memory usage.

2. Related Works

In the literature, many terms refer to the concept of self-reconfiguration. In several works on wireless networks and mobile robots [17] the term used is *self-organization* . This term is also used to express the partitioning and clustering of ad-hoc networks or wireless networks to groups called cliques or clusters. The objective is to determine the various spots where the organization is carried by the cluster head that has more amount of energy. Also the self-organization term can be found in protocols for sensor networks to form a sphere or a polygon from a center node [18, 19, 20]. The term *redeployment* is also another term to address self-reconfiguration for sensor networks [21, 22]. In [23] an efficient protocol addresses the problem of autonomous deployment of mobile sensors that need to cover predefined positions with a connectivity constraint.

For the self-reconfiguration with robots or microrobots there are the protocols [16, 24] where the desired configuration is grown from an initial seed module. A generator uses a 3D CAD model of the desired configuration and outputs a set of overlapping blocks which represent this configuration. In the second step this representation is combined with a control algorithm to produce the final self-reconfiguration algorithm.

A growing number of researchers are working on reconfiguration of microrobots using centralized algorithms. Among them, we find the proposed control algorithms for self-assembly and/or reconfiguration with a centralized manner, see for instance [25]. Other approaches give each node a unique ID and predefined positions of the target shape, see for instance [26]. The disadvantage of these methods is the centralized computing and the need for nodes identification. More distributed approaches that need the map of the target shap in [27, 28, 29, 30]. In simulation, the authors in [24, 31] have demonstrated algorithms for self-reconfiguration of cubic units based on gradients and cellular automata. Bojinov and al. [32] have shown how a simulated modular robot (Proteo) can self-configure into useful and emergent morphologies when the individual modules use local sensing and local control rules.

In [33] the authors developed a centralized algorithm for reconfiguration (with predefined positions of the target chain) of an initial chain configuration into another chain configuration and then from a straight chain into an arbitrary goal that fulfills certain admissibility requirements [34]. The distributed version of this algorithm was given in [35]. Recent work in [36] demonstrated a time complexity of $O(n)$ for probabilistic reconfiguration a systems of hexagonal metamorphic robots for single-move algorithms, in which at most one module can move in a time step.

Claytronics, which stands for *Clay-Electronics* is the name of a recent robotics project by Carnegie Mellon University and Intel Corporation, in which nanoscale robots called

Catoms (Claytronics Atoms) are assembled to form larger objects. It is designed by Researchers from Carnegie Mellon University and Intel Corporation that have been working on this project where the idea is to have hundreds of thousands of nanoprocessors (microrobot) form together to create new solid objects in any shape or size. The challenges are vast, but once it becomes a reality it will change the world forever. Much like the cells in a body or complex organism, each small member of the whole is committed to doing its own part and communication between parts results in a unified form.
Many works have been done on the Claytronics project. In [37], the authors propose a metamodel for the reconfiguration of catoms starting from an initial configuration to achieve a desired configuration using *creation* and *destruction* catoms primitives. The authors use these two functions to ignore the inability of motion of Catoms in the presence of neighbors that can be considered as barriers. In [10] the authors present a scalable distributed reconfiguration algorithm with the Hierarchical Median Decomposition, to achieve arbitrary target configurations with a map and without a global communication. Another scalable algorithm can be find in [11]. In [9] a scalable protocol for Catoms self-reconfiguration is proposed, written with the MELD language [8] [38] and using the creation and destruction primitives. In all these works, all Catoms know the correct positions composing the target shape at the beginning of the algorithm and each node is aware of its current position. Other solutions without predefined positions of the target shape in [39, 40, 41, 42]. In these protocols, the starting physical topology is a chain of MEMS microrobots.

3. Contributions and comparison with literature works

In this paper we present a map-less and energy-efficient distributed self-reconfiguration protocol. The proposed algorithm takes into account the technological constraints on MEMS microrobots related to memory and energy limitations. A self-reconfiguration with a map of the target shape is not memory-efficient, because in order to cover a target shape consists of a set of positions, it will be required to divide this target shape to very small units according to the size of the microrobots which is very small, which will give millions or billions of positions, therefore each node should have a memory capacity of millions or billions of positions, hence the importance of a reconfiguration protocol without map of the target shape. That is, in the literature works when a self-reconfiguration process aims to reach a given target shape composed of a set of P positions (like a pixels for a given picture), each microrobot records all the P positions. This is neither efficient nor scalable since we address here configurations with millions of nodes have a low-memory capacity, and millions of final positions because the MEMS nodes have a very small size.
In this work, we propose a new efficient approach for self-reconfiguration of MEMS microrobots where nodes do not record any position and where the target shape is built incrementally. Each node in the current increment acts as a benchmark point for other nodes to form the next increment. The proposed model makes the assumption that each node can obtain the state of its physical neighbors to achieve self-reconfiguration for distributed MEMS. Using these states, nodes collaborate and help each other without need for a global information. Contrary to existing works, our algorithms do not need to know the map of the target shape (i.e. coordinates of the microrobots), consequently memory usage is dramatically reduced. The presented approach has the advantage of not requiring the node knowledge of its own position either. Such knowledge would need a positioning technology which would require a lot of energy or a multilateration method [43, 44] which is not sufficiently precise, this imprecision can lead to incorrect behaviors of the self-reconfiguration applications. In all applications of self-reconfiguration, knowing the exact position of every node is an important factor in receiving expected programmed behavior. This paper is the first to provides a self-reconfiguration standalone and portable for any starting shape, because it is independent of the map that builds the target shape. In this solution, the self-reconfiguration is achieved using the help and the collaboration between nodes, so to make intelligent algorithm by analyzing the characteristics of the target shape.

We propose a scalable protocol for node's reconfiguration to optimize communication for MEMS nodes. This protocol consists of two steps *Initiator Election* and *Shape Shifting*. The initiator election algorithm aims to choose the best initial node that will initializes the algorithm Shape Shifting, the criteria of choice is the effect on the amount of movement on the other nodes. The initiator selection is made on the basis of the predicted amount of movements of the other nodes, needed to achieve the target shape. The objective is then to select the node leading to the minimum amount of movements. We try to apply a principle of "*the target shape arrives to nodes*" and reach the target shape with a minimum amount of movements. The shape shifting algorithm aims to convert the initial form with an incremental process starting with an initiator node (representing a punctual square) and adding every time a new layer composed of a number of nodes equal to the previous layer plus two nodes.
The remainder of this paper is organized as follows: Section 4 discusses the model and terminology. Section 5 discusses the proposed protocol and analyzes its characteristics. Section 6 discusses the simulation results based on the Dynamic Physical Rendering Simulator (DPRsim)[45]. Finally, section 7 summarizes our conclusions and illustrates our suggestions for future work.

4. Problem Statement

Model and definitions : Within Claytronics, a Catom that we call in this paper a node (see Figure 1) is modeled as a sphere which can have at most six 2D-neighbors without overlapping. Each node is able to sense the direction of its physical neighbors (east (E), west (W), north-east (NE), south-east (SE), south-west (SW) and north-west (NW)). In this work, the starting topology can be an arbitrary connected physical topology with n nodes linked together. A node A is in neighbor's list of node B if A touch physically B. Communications are only possible through contact, which means that only neighbors can have a direct communication.

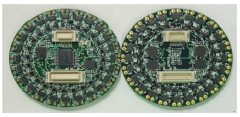

Figure 1: Two catoms

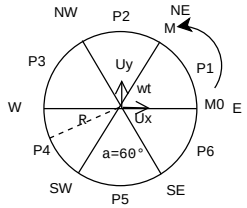

Figure 2: Node modeling, in each movement the node travels the same distance

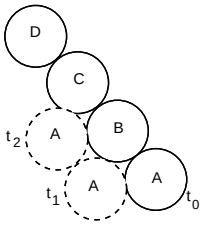

Figure 3: Message transmission, there will be message exchange if the node needs to know the state of a non-neighbor node

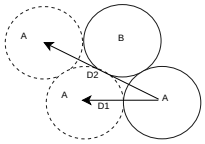

Figure 4: Traveled distance in one movement = $2R$, the node A travels 2xR in one movement

To calculate the number of movements we define the following:
Consider Figure 2 which represents a microrobot. We say that a microrobot has done a single movement if the distance between its former position and its new position is exactly twice the radius $D1 = 2R$. For example, if the node is in a position at a distance $D2$ (see Figure 3) from the former position it has done two movements. We have 360° can be divided to six equal angles each one has 60°, since the perimeter at an angle a is $P_a = \pi\ Ra/180$ and $P = 2\pi R$ we find $P1 = P2 = P3 = P4 = P5 = P6$, this means that the node can have without overlapping at most six neighbors and in each movement the node travels Ra (with $a = 60°$) from $m0$ to m.

Consider the connected undirected graph $G = (V, E)$ modeling the network, where $v \in V$, is a node that belongs to the network and, $e \in E$ a bidirectional edge of communication between two physical neighbors. For each node $v \in V$, we denote the set of neighbors of v as $N(v)$. Each node $v \in V$ knows the set of its neighbors in G, denoted $N(v)$. We define *Spanning tree*: is a tree composed of all $v \in V$ without any cycle. In the spanning tree, a node is either a child or a parent. Leaves are nodes without children.
To facilitate the explanation of the algorithms we model the network as follows:

- A node N in a matrix $J * I$ where we can find an empty box(j, i)(there is no node in this box), the node N in row i and column j called $N(j, i)$, see Figure 5.

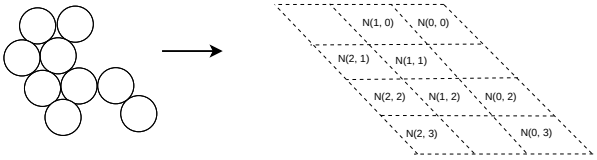

Figure 5: An example of a network modeled into a matrix

- We call *best width* of a node $N(j, i)$, the value $cntW = J - j$ and *best height* the value $cntH = I - i$.
- We call *absolute best initiator* the node $N(0, 0)$, knowing that counting is from bottom to down for the row i and from right to left for column j.

We assume (and according to the tools of simulation) that message exchanging between two physical neighbors is carried without time complexity (0 rounds). This is according to the simulation tools where the node can see the state of its physical neighbor. However, when a node consults the state of not neighboring node then the consultation leads to message exchange that takes time and thus involves the decision delay. Figure 3 illustrates those cases:

- At t_0: Node A needs to know the state of the node B to move to the new position. This motion is done by a simple read action (without message exchanging).
- At t_2: If A is in the new position and needs to know the state of the node D, this last sends a message to C containing its state, after which C forwards the message to A. There is a message exchange; A must wait two rounds for the input to decide.
- If at t_0 or at t_1 a message has been sent from D to C, A at t_2 can have the state of D with a simple consultation of C's state, without message exchange.

Objective: As described earlier, we propose reconfiguration protocol without map (without predefined positions of the target shape) optimizing the amount of displacements. This protocol has another goal at the same time which is

to optimize the physical topology. That is, the chain is the configuration (physical topology) that represents the worst complexity for messages exchanging, within the chain if the two nodes in both ends of the chain exchange message the complexity will be of $O(n)$, where n is the network size. The physical configuration that represents the best complexity in terms of messages exchanging is a square with $O(\sqrt{n})$ in the worst case. The objective of our protocol is also to improve the propagation procedure, fault tolerance and convergence. This protocol will convert any configuration of a connected network of microrobots in a square configuration.

5. Proposed Protocol

The protocol consists of two algorithms, *initiator election* and *shape shifting*. The initiator election aims to choose the best initiator, the criteria of choice is the effect on the amount of movement on the other nodes, we try to apply a principle of *the target shape arrives to nodes* and reach the target shape with a minimum amount of movement. The shape shifting aims to convert the form with an incremental process, starting with a node that assumed a square and each time we add a new layer with number of nodes equal to previous layer plus two nodes. The nodes of each layer added change their state and become stable nodes. The shape shifting algorithm shows how the problem of synchronization in state changing was solved using states and messages.

5.1. Initiator election

Overview: The objective of *initiator election* procedure is to identify simply and quickly the starting point of the shape-shifting procedure in order to guaranty a minimum amount of movements required to reach the target shape. The idea is twofold. First the selected initiator node represents the top-right corner of the targeted square, restricting the redeployment surface to a quarter of the plan. Secondly, the initiator node is selected in order to minimize the number of nodes out of this quarter of plan. The combined action of those two principles makes that almost all nodes are in the same area and move in this area reducing therefore the needed displacement amount. Furthermore the impact of the *initiator election* procedure on the shape-shifting displacements will be discussed and analyzed in the experimental section of this paper. In this protocol, the nodes will be organized only in quarter of the plan, e.g. with the example in figure 6), nodes movement will be carried only in the area A1, and long distances are avoided. With the example of figure 6 the best initiator is the node 1 because no node is out of the plan if the node 1 has been chosen. If this node is not present (figure 7), the best initiator will be the node 4, because there is a probability of $2/n$ of node's existence out of the area A1-a1, in this case 2 nodes (the node 2 and the node 3) do not belong to the plan A1-a1. Similarly, if the node 4 is not present the best initiator will be the node 2, and so on.

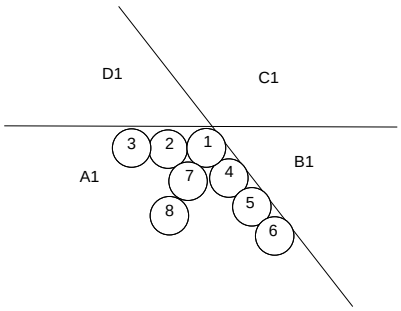

Figure 6: Nodes move in the area $A1$

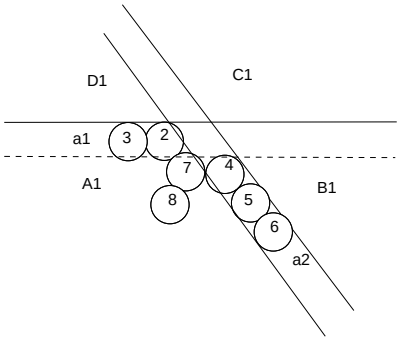

Figure 7: Nodes move in the area $A1 - a1$, nodes that are outside this area will join it, as the case of node 2 and 3

With the initiator election algorithm, all nodes converge toward the same best initiator. The algorithm consists in two steps. In step 1 called *coordinates finding* a distributed algorithm is run. In step 2 a heuristic algorithm called *best initiator finding* is executed locally on each node using the information obtained by coordinates finding step. In step 1, each node calculates its best width and its best height values and saves those of the other nodes. The aim of the initiator election algorithm is to find the node that will initialize the shape shifting algorithm.

5.1.1. Coordinates Finding

This section presents the Coordinates Finding algorithm. The coordinates finding algorithm (presented hereafter) runs in rounds. At each round, the satisfied predicates are chosen to run. In this section, each node finds its best width ($cntW$) and best height ($cntH$), and it saves those other nodes. The information obtained in this section will be used in the next step best initiator finding algorithm. At the end of the coordinates finding algorithm each node has registered a list L containing all the coordinates of nodes by choosing for each id received in feedback ($receiveRESP_v(id, cntH, cntW)$) the $MAX(cntH)$ and $MAX(cntW)$.

Description of the algorithm coordinates finding

The predicate $BroadcastMSG_v(id, 0, 0)$ is true for each node connected to the network. The node sends the message $(id, 0, 0)$ to all its neighbors. With the predicate $ReceiveMSG - X_v(id, cntH, cntW)$ the neighbors receive

Variables and Predicates

- initiator$_v()$: each node initiates the algorithm.
- broadcastMSG$_v(id, 0, 0)$: each initiator sends a message containing its identity id, a variable to count the best height $cntH = 0$, and a variable to count the best width $cntW = 0$.
- receiveRESP$(id, cntH, cntW)$: the node receives the message $(id, cntH, cntW)$ in feedback if it was the first message with same parameters $id, cntH, cntW$.
- receiveMSG$-X_v(id, cntH, cntW)$: with $X \in \{NW, NE, E, SE, SW, W\}$. The node receives the message in brodcast if it was the first message with identity id.
- visitedB$_v(id)$: *true* if the node already received a message in broadcast with the same id.
- visitedF$_v(id, cntH, cntW)$: visited on feedbak (response), *true* if the node received on feedabk the same id with the same $cntH$ and $cntW$.
- broadcastMSG$_v(id, cntH, cntW + 1)$: increment $cntW$ and broadcast the message received by the node in E direction.
- broadcastMSG$_v(id, cntH + 1, cntW + 1)$: increment $cntH$ and $cntW$ and broadcast the message received by yhe node in NE direction.
- broadcastMSG$_v(id, cntH + 1, cntW)$: increment $cntH$ and broadcast the message received by the node in NW direction.
- broadcastMSG$_v(id, cntH, cntW - 1)$: decrement $cntH$ and broadcast the message received by the node in W direction.
- broadcastMSG$_v(id, cntH - 1, cntW)$: decrement $cntH$ and broadcast the message received by the node in SE direction.
- broadcastMSG$_v(id, cntH - 1, cntW - 1)$: decrement $cntH$ ans $cntW$ and broadcast the message received by the node in SW direction.
- broadcastRESP$_v(id, cntH, cntW)$: nodes that do not have neighbors in the direction E, SW and SE, and others that do not have neighbors in the directions W, SE and SW will begin the feedback.

- receiveRESP$_v(id, cntH, cntW)$: the node can broadcsat the feedback if it was the first message with the same id, $cntH$ and $cntW$.
- broadcastRESP$_v(id, cntH, cntW)$: the node bradcasts the feedback to the other nodes until the message arrives to the node id.

Predicates checked only in the first round

$$\text{initiator}_v() \equiv Connected_v.$$
$$\text{broadcastMSG}_v(id, 0, 0) \equiv initiator_v().$$

Predicates checked in each round

$$\text{receiveMSG-NE}_v(id, cntH, cntW) \equiv (Vne(v)) \wedge (\neg visitedBv(id)).$$
$$\text{receiveMSG-NW}_v(id, cntH, cntW) \equiv (Vnw(v)) \wedge (\neg visitedBv(id)).$$
$$\text{receiveMSG-SE}_v(id, cntH, cntW) \equiv (Vse(v)) \wedge (\neg visitedBv(id)).$$
$$\text{receiveMSG-SW}_v(id, cntH, cntW) \equiv (Vsw(v)) \wedge (\neg visitedBv(id)).$$
$$\text{receiveMSG-E}_v(id, cntH, cntW) \equiv (Ve(v)) \wedge (\neg visitedBv(id)).$$
$$\text{receiveMSG-W}_v(id, cntH, cntW) \equiv (Vw(v)) \wedge (\neg visitedBv(id)).$$
$$\text{visitedB}_v(id) \equiv receiveMSG - X_v(id, -, -).$$
$$\text{broadcastMSG}_v(id, cntH, cntW + 1) \equiv receiveMSG - E_v(id, cntH, cntW).$$
$$\text{broadcastMSG}_v(id, cntH + 1, cntW + 1) \equiv receiveMSG - NE_v(id, cntH, cntW).$$
$$\text{broadcastMSG}_v(id, cntH + 1, cntW) \equiv receiveMSG - NW_v(id, cntH, cntW).$$
$$\text{broadcastMSG}_v(id, cntH, cntW - 1) \equiv receiveMSG - W_v(id, cntH, cntW).$$
$$\text{broadcastMSG}_v(id, cntH - 1, cntW) \equiv receiveMSG - SE_v(id, cntH, cntW).$$
$$\text{broadcastMSG}_v(id, cntH - 1, cntW - 1) \equiv receiveMSG - SW_v(id, cntH, cntW).$$
$$\text{visitedF}_v(id, cntH, cntW) \equiv ReceveRESP_v(id, cntH, cntW).$$
$$\text{broadcastRESP}_v(id, cntH, cntW) \equiv (\neg Ne(v)) \wedge (\neg Nsw(v)) \wedge (\neg Nse) \wedge \neg visitedF_v(id, cntH, cntW).$$
$$\text{broadcastRESP}_v(id, cntH, cntW) \equiv (\neg Nw(v)) \wedge (\neg Nse(v)) \wedge (\neg Nsw) \wedge \neg visitedF_v(id, cntH, cntW).$$
$$\text{receiveRESP}_v(id, cntH, cntW) \equiv \neg visitedF_v(id, cntH, cntW).$$
$$\text{broadcastRESP}_v(id, cntH, cntW) \equiv receiveRESP_v(id, cntH, cntW).$$

The Coordinates Finding Algorithm

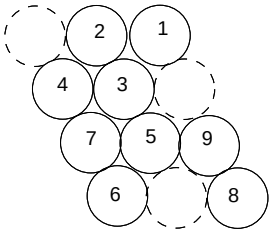

Figure 8: An example of network with nine nodes

the message where the guard $visitedB_v(id)$) ensures that the node receives a message in broadcast of this type only once per each *id*. After receiving this message, the node depending on the direction of reception it increments, decrements or it does nothing on the parameters $cntH$, $cntW$. To calculate the best width: for each message received by the E or NE directions the node increments the parameter $cntW$. If the message was received by the W directions or by the SW directions, then the node decrements $cntW$. The same procedure is carried for the best height, whenever the message was received by the NE or NW directions, the node increments $cntH$, and the node decrements $cntH$ if the message was received by SE or SW directions. Nodes that do not have neighbors in the E, SW and SE directions, and other nodes that do not have neighbors in the W, SE and SW directions will begin the feedback (response message) just after receiving the broadcast message for an identity *id*, these nodes broadcast the message $BroadcastRESP_v(id, cntH, cntW)$, and neighbors which receive this message will rebroadcast it, the guard $visitedF_v(id, cntH, cntW)$ is used to avoid broadcasting the same message with the same parameters $cntH$ and $cntW$.

Example

We take the example with the node 1 in Figure 8 knowing that other nodes will do with the same procedure. The node 1 sends to its neighbors 2 and 3 the message (1, 0 ,0) with predicate $broadcastMSG_v(id, 0, 0)$ and waits for answers with the predicate $receiveRESP(id, cntH, cntW)$. At the reception of this message, the node 2 increments with $broadcast_v(id, cntH, cntW + 1)$, and node 3 increments and broadcasts with $broadcast_v(id, cntH+1, cntW+1)$. So the message becomes (1, 0, 1), (1, 1, 1) for the node 2 and 3 respectively. For the node 4, at the reception of the message (1, 0, 1) sent by the node 2 or (1, 1, 1) sent by the node 3, it increments and broadcats with $broadcast_v(id, 1cntH + 1, cntW + 1)$ (if the message was sent by node 2) or with $broadcast_v(id, cntH, cntW + 1)$ (if the message was sent by node 3), the message becomes (1, 1, 2). The node 7 at the reception of (1, 1, 2) sent by node 4 or (1, 1, 1) sent by node 3 increments with the suitable predicate depending on the direction of reception, the message becomes (1, 2, 2), the node 7 broadcasts this new message.

Coming back to the node 5, this node at the reception of (1, 1, 1) sent by node 3, the node 5 increments and broadcasts (1, 2, 1) with the predicate $BroadcastMSG_v(id, cntH + 1, cntW)$, the node 6 at the reception of (1, 2, 2) sent by the node 7 or (1, 2, 1) sent by 5, it increments with the suitable predicate, the message becomes (1, 3, 2). Similarly, the node 9 receives and decrements with the predicate $broadcastMSG_v(id, cntH, cntW - 1)$ and the message becomes (1, 2, 0), the node 9 sends the message to the node 8, this last increments to (1, 3, 0). The nodes 6 and 8 can start the feedback procedure with sending the message (1, 3, 2) and (1, 3, 0) to others nodes (with $broadcastRESP_v(id, cntH, cntW)$), each node receives only once the same message and takes always the max of $cntH$ and $cntW$ e.g. the node 5 receives in feedback (1, 3, 0) and (1, 3, 2) in this case the node 5 saves in its local memory the message (1, 3, 2).

The list Ln presents the list of node's coordinates saved by each node at the end of the algorithm:
$Ln = \{(1,3,2),(2,3,1),(3,2,1),(4,2,0),(5,1,1),(6,0,0)\} \cup \{(7,1,0),(8,0,2),(9,1,2)\}$.

Lemma 1. *Since each node saves for each id, the values cntH and cntW , the algorithm coordinates finding requires* $3n$ *of memory for each node.*

Lemma 2. *Since the chain shape represents the worst physical topology for many distributed algorithms in terms of fault tolerance, propagation procedures and convergence, and since nodes initialize the algorithm in parallel, the lengthy communication will be between the two ends of the chain with* $2n$ *messages, n messages for the broadcast and n message for the response (feedback).*

5.1.2. Best Initiator Finding

The best initiator finding algorithm (presented hereafter) is executed by each node in the networks. This algorithm uses the information acquired in coordinates finding algorithm to elect the initiator that will initiate the shape shifting algorithm.

The absolute best initiator if exists has the coordinate(J, I), with $J = MAX(cntH)$ and $I = MAX(cntW)$, with $cntW$ and $cntH \in Ln$, it takes the coordinate (0, 0). Then, each node takes the coordinates $(J - cntH, I - cntW)$ and does the same subtraction for all coordinates stored in its local memory. So the list will have this forme $L = \{(x1, y1), (x2, y2), ..., (xn, yn)\}$. We define the function $ID(x, y)$ that gives the *id* of the node having the coordiantes (x, y).

Description of the algorithm best initiator finding

The absolute best initiator, if it exists, has the coordinates (0, 0). At the beginning, each node checks if the coordinates (0, 0) is in its local memory. If so, the node ID(0, 0) declares itself as an initiator, and the other nodes acknowledge it. If there is no absolute best initiator, the candidate nodes will be in the list LC, these candidates are neighbors of the candidates (that does not exist) were in LC in the previous round. For each candidate in LC the number of nodes out the plan Nli is calculated, and the node with minimum Nli will be elected. If this elected node does not exist by checking in the list L, a new LC is considered in the next round and so on.

Algorithm for each node

Variables

- list: $A, li, LC, R1, R2$.
- integer: Nli initialized to 0, represents the number of nodes out the area if the node having li as coordinate is chosen.

$A = \{(0,0)\}$;
if $\exists a = (x,y) \in A \wedge a \in L$ **then**
end of the algorithm, the initiator is $ID(a)$; **else**
begin
$LC = \oslash$;
for all $a = (x,y) \in A, LC = LC \cup \{(a.x+1, a.y) \cup (a.x+1, a.y+1) \cup (a.x, a.y+1)\}$;
end for all
$LC = \{l1, l2, l3, ..., ln\}$, with $\forall li \in LC, li \notin A$;
for each $li = (x,y)$ of LC the node computes NLi:
begin
calculate the list $R1$ and $R2$ for li ;
Let $X = MAX(xi, i \in \{1..n\}), with\,(xi, yi) \in A$
Let $Y = MAX(yi, i \in \{1..n\}), with\,(xi, yi) \in A$
$R1 = \{(X, y1), (X, y2), ..., (X, yn)\}$. With: $y1 \neq y2 \neq, ..., \neq yn$, $yi < y$ and $R1 \subseteq A$

$R2 = \{(x1, Y), (x2, Y), ..., (xn, Y)\}$. With: $x1 \neq x2 \neq, ..., \neq xn$, $xi < x$ and $R2 \subseteq A$

for all $xi \in R1$:
For $m = xi + 1$ to $JNLi++$;
for all $yi \in R2$:
For $m = yi + 1$ to $INLi++$;
end
$BEST = li$ having$(MIN\,NLi)$;
If $BEST \in L$ **then**
End of the algorithm, the best is $ID(BEST)$.
else
begin
$A = A \cup li$;
Repeat the algorithm;
end
end

The Best Initiator Finding Algorithm

5.2. Shape Shifting

Description and analysis of the algorithm

The Shape Shifting algorithm (presented hereafter) runs in rounds. At each round, the satisfied predicates are chosen to run. To describe the algorithm we define the *first row* all nodes having the state FR which are at the left of the initiator and the *first column* all nodes having the state FC which are at the south east of the initiator. A*new layer* is constructed of a *new column* and a *new row* as shown in Figure 10. A node becomes *stable* once it has state the state *well*, because it belongs to a position in the target shape.

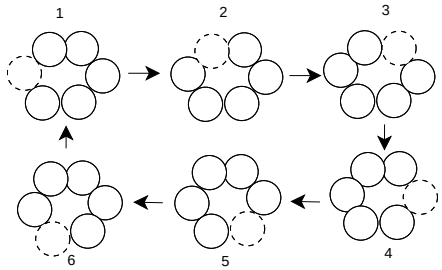

Figure 9: Nodes cannot apply the principle of *node follows its neighbor* to ensure the connectivity and to avoid losing nodes, because of a cycle. Moreover, nodes may enter in collusion state when they move at the same time to the same place. Therefore, nodes may be move in a boucle and do not converge towards the target shape. So the solution is to use a spanning tree: rooted by the initiator and the node follows its parent, and after each movement the parent waits for one child

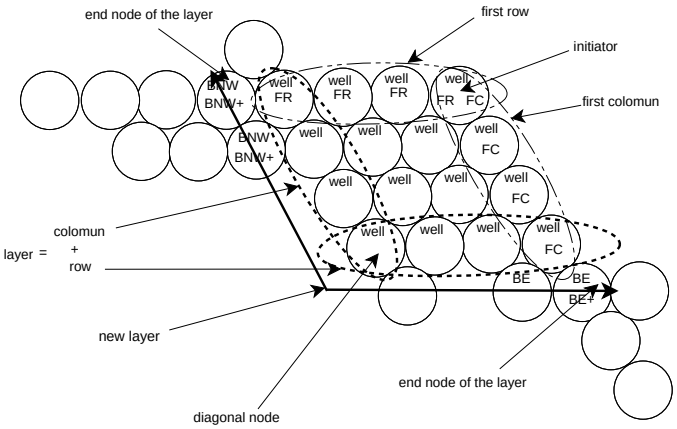

Figure 10: An instance of nodes with their states

The algorithm runs in rounds. At each round, the satisfied predicates are chosen to run. The distributed algorithm seeks the desired form by using an incrementally process. In a completed increment, the nodes that build it belong already to the form; these nodes will help neighbor nodes and future neighbor nodes to get correct positions. That is, the node makes decisions and actions depending on neighbors states.

The initiator takes the state *well* and becomes stable with the predicate (2), and each node will be stable after having the state *well*. With predicate (1), the initiator also initializes the spanning tree construction, the spanning tree is used to provide network connectivity and avoid losing nodes, since communication is possible only through contact. If a node has no neighbors at the beginning it cannot find or join another node. The use of the spanning tree is also to avoid infinite movement. Indeed, if there is a cycle in the network there will be an infinite movement, because if we assume to ensure the network connectivity a node follows the neighbor node that has moved in the previous, we can find a case where the nodes move in a cycle, as the case in Figure 9, by definition the tree is a connected network without cycle.

Variables and Predicates

- parent(v, v): the initiator(the root of the tree) is a parent of itself.
- isLeaf(v): the node v is a leaf, not a parent.
- parent(v, u): the node v is parent of its neighbor u.
- state$_v(X)$: v has the state $X \in \{wel, bad, FR, FC, BE, BE+, BNW, BNW+\}$, we note that the node cannot take the state *well* and *bad* at the same time. At the beginning all nodes are initialized with *bad* state. We note that the node loses its state $BE+$ and $BNW+$ if it moved and it reserves the other states once obtained.
- moveAround$well_v(u1, P_X)$: v moves around $u1$ until this last becomes in the direction NW relative to v.
- moveTo$_v(u1, pos_{u1})$: the node v moves to the old position of its parent $u1$.
- r:an integer denotes the current round of the algorithm for the node.

1. parent$(v, v) \equiv initiator_v()$.
2. state$_v(well) \equiv initiator_v()$.
3. isLeaf$(v) \equiv (\nexists u, parent(u, v)) \wedge state_v(bad)$.
4. parent$(v, u) \equiv (parent(w, v), u \neq w) \wedge (N(v) = u) \wedge state_u(bad) \wedge (\nexists z \in N(v), parent(v, z))$.
5. state$_v(FR) \equiv initiator_v() \vee (Ne(v) = u \wedge state_u(FR) \wedge state_v(well))$.
6. state$_v(FC) \equiv initiator_v() \vee (Nnw(v) = u \wedge state_u(FC) \wedge state_v(well))$.
7. state$_v(BE) \equiv (Nnw(v) = u \wedge state_u(well) \wedge state_u(FC)$.
8. state$_v(BE) \equiv (Ne(v) = u1 \wedge state_{u1}(bad) \wedge state_{u1}(BE)) \wedge (Nne(v) = u2 \wedge state_{u2}(well)) \vee ((Nse(v) = u1 \wedge state_{u1}(bad) \wedge state_{u1}(BE)) \wedge (Ne(v) = u2 \wedge state_{u2}(well)))$.
9. state$_v(BNW) \equiv (Ne(v) = u \wedge state_u(well) \wedge state_u(FR)$.
10. state$_v(BNW) \equiv (Nnw(v) = u1 \wedge state_{u1}(BAD) \wedge state_{u1}(BNW)) \wedge (Ne(v) = u2 \wedge state_{u2}(well)) \vee ((Nw(v) = u1 \wedge state_{u1}(bad) \wedge state_{u1}(BE)) \wedge (Nne(v) = u2 \wedge state_{u2}(well)))$.
11. state$_v(BE+) \equiv (Nnw(v) = u \wedge state_u(well)) \wedge \neg IsLeaf(v)$.
12. state$_v(BE+) \equiv (Ne(v) = u1 \wedge state_{u1}(bad) \wedge state_{u1}(BE+) \wedge state_{u1}(BE)) \wedge (Nnw(v) = u2 \wedge state_{u2}(well)))$.
13. state$_v(BNW+) \equiv (Ne(v) = u \wedge state_u(well)) \wedge \neg IsLeaf(v)$.
14. state$_v(BNW+) \equiv (Nnw(v) = u1 \wedge state_{u1}(bad) \wedge state_{u1}(BNW+) \wedge state_{u1}(BNW)) \wedge (Ne(v) = u2 \wedge state_{u2}(well)))$.
15. state$_v(well) \equiv (Nnw(v) = u1 \wedge state_{u1}(BNW) \wedge Nse(v) = u2 \wedge state_{u2}(BE)) \vee (Nw(v) = u1 \wedge state_{u1}(BNW) \wedge Ne(v) = u2 \wedge state_{u2}(BE))$.
16. state$_v(well) \equiv state_v(BEW) \wedge Nse(v) = u1 \wedge state_{u1}(BE)) \vee (state_v(BE) \wedge Nw(v) = u2 \wedge state_{u2}(BNW))$.
17. moveAround$well_v(u1, P_{nw}) \equiv state_v(bad) \wedge ((Nne(v) = u1 \wedge Nnw(v) = u2 \wedge state_{u1}(well) \wedge state_{u2}(well))$.
18. moveAround$well_v(u1, P_e) \equiv state_v(bad) \wedge ((Nne(v) = u1 \wedge Ne(v) = u2 \wedge state_{u1}(well) \wedge state_{u2}(well))$.
19. moveAround$well_v(u1, P_{ne}) \equiv state_v(bad) \wedge state_v(BNW+) \wedge (Ne(v) = u1 \wedge state_{u1}(well))$.
20. moveAround$well_v(u1, P_{ne}) \equiv state_v(bad) \wedge state_v(BE+) \wedge (Nnw(v) = u1 \wedge state_{u1}(well))$.
21. moveAround$well_v(u1, P_{ne}) \equiv state_v(bad) \wedge ((Nne(v) = u1 \wedge Ne(v) = u2 \wedge state_{u1}(well) \wedge state_{u2}(BE)) \wedge state_v(BE+)$.
22. moveAround$well_v(u1, P_{ne}) \equiv state_v(bad) \wedge ((Ne(v) = u1 \wedge Nnw(v) = u2 \wedge state_{u1}(well) \wedge state_{u2}(well)) \wedge state_v(BEW+)$.
23. moveAround$well_v(u1, P_{se}) \equiv state_v(bad) \wedge (Nsw(v) = u1 \wedge state_{u1}(FR))$.
24. moveAround$well_v(u1, P_w) \equiv state_v(bad) \wedge (Nsw(v) = u1 \wedge state_{u1}(FC))$.
25. moveAround$well_v(u1, P_e) \equiv state_v(bad) \wedge (Nse(v) = u1 \wedge state_{u1}(FR))$.
26. moveAround$well_v(u1, P_{nw}) \equiv state_v(bad) \wedge (Nsw(v) = u1 \wedge state_{u1}(FC))$.
27. moveTo$_v(u1, pos_{u1}) \equiv state_v(bad) \wedge (\exists x Nx(v) = u1 \wedge parent(u1, v) \wedge, r = r - 1) \wedge (\forall x, \neg \exists Nx(v) = u, state_u(well)) \wedge (\exists x Nx(v) = u2 \wedge parent(v, u2))$.
28. moveTo$_v(u1, pos_{u1}) \equiv state_v(bad) \wedge (\exists x Nx(v) = u1 \wedge parent(u1, v) \wedge isLeaf(v), r = r - 1)$.

Therefore, with the tree, one of the child nodes follows its parent. After making a movement, a parent cannot make another movement except after being followed by a child and becomes a parent in the current round, with a principle of *The child follows its parent and the parent waits for its child.* With the predicate (4) other nodes take a child, except the leaves which cannot be parents, because all neighbors are parents (3).
With the predicate (5) the nodes of the first row take the state FR and with (6) the nodes of the first column take the state FC. These two last states are used to manage nodes outside the surface of the final square, and to bring them into the surface with predicates (23), (24), (25) and (26). The state BE is used to fill a new row starting from right to left (the nodes take the state BE from right to left), the node that has a neighbor in NW direction having the state FC is the first in a new layer that takes the state BE (7), this node does not move unless it has a child in the tree (a child necessarily has no state except the state *bad*) because this node begins the construction of a new row of the new layer. After, a node takes the state BE if it has a neighbor in the E direction having the state BE, this is with predicate (8). The state BNW is used to populate a new column starting from top to bottom (the nodes take the state BNW from top to bottom) with both predicates (9) and (10). A node that has the state BE or BNW never moves unless it has a child, therefore the state $BE+$ or $BNW+$ or if it received a message from a node that has a child but cannot move.
Thus, with (11) a node takes the state $BE+$ if two conditions are met: this node must be in the layer being built (new layer) which can be checked with the neighbor in direction E/NW, this node must be in a *well* state and it has a child, if these two conditions are hold, and if it finds a space to move in the direction W/SE it makes the motion because one of its neighbor or its child will take its old position with predicates (27) and (28). The node can take the state $BE+$ after receiving a message with predicate (12) informing that there is a node or nodes having a child but they cannot move. If this node has an empty space, it moves around a node having the state *well* with the predicate (20). Similarly, the node having state BNW can take the state $BNW+$ if it has a child (13), and it moves in the SE direction if this space is empty with the predicate (19), if this space is filled, the message will be transmitted with the predicate (14).
The change of state to *well* is done with the predicates (15) and (16).With these two predicates, the node changes its state after being sure that the new layer is completely filled. So with (15) the node having the neighbor in the NW direction that has the state BNW and a neighbor in the SE direction having the state BE takes the state *well*, because the states BE/BMW are propagated from nodes in the current layer having the state FL/FC, so if the node has two neighbors with these two states it is sure that the new layer (row and column) is filled and the change of state to *well* is possible. The predicate (16) is similar to (15) except that, in (16), the node checks if its state is BNW or BE. These procedures solve the synchronization problem in state changing to *well*, because these nodes (nodes building the new layer) do not have a global view of the network and they cannot know when the new layer is completely filled so they can change the state.
At the beginning, a new layer must be built from right to left, for a row, and, from top to bottom for a column. The node in a new layer and before having any state it moves to right using the predicate (17) until it has a neighbor in the direction NW having the state FC or until it has a neighbor in the direction E having the state BE. Similarly to build the column node moves up with the predicate (18). With the predicates (19) and (20) the nodes which are building the new layer (having neighbors with *well* state) move to left /bottom only if it has a child or after receiving a message $BNW+/BE+$ from a node in the new layer having a child. The predicates (21) and (22) allow nodes in a new layer to move from a row to a column or vice versa. The predicate (21) allows moving around the diagonal node and going build a row. The predicate (22) allows nodes to go from row to column to build a new column. The predicates (23), (24), (25), and (26) allow to nodes which are outside the surface to enter in it. These nodes move around a node having the state FR or FC to enter the surface and contribute to build a new layer. The predicates (27) and (28) help to ensure network connectivity using the tree. With the predicate (27), a node moves to the position of its parent after the displacement of one of its child to its former position, with the predicate (28) a leaf follows its parent.

5.3. Message transmission analysis

In this section we analysis the number of messages required for the nodes to change their states to *well*, so each node be aware that it is in the final form.

5.3.1. In the best case

Lemma 3. *In the best case, the shape shifting algorithm needs* $2n - 2\sqrt{n} + O(n) - 1$ *messages.*

The justification is the following. The root that represents itself a layer changes its state without using a message, because it does not need to know any state from its neighbors. The best case for the number of message exchange is when the starting the shape is already a square because no node will do the movement, and it needs only to change its state to well and become a stable node. In this case, the message transmission can be done in parallel. In a current layer (layer being built), the two end nodes of the layer wait for the transmission of the state $BE(BMW)$*. Since the transmission takes place in parallel, each end node must wait the state* $BE(BNW)$ *from the other end node of the layer. The message* $BE(BNW)$ *will visit all nodes of the current layer. We find that* $n-1$

messages are required.
The nodes of the next layer cannot do anything since the nodes of the current layer do not have the state well so they wait the change of state to well of nodes of the current layer. Nodes of the next layer have therefore to wait for number of message equal to the size of the current layer, except nodes of the last layer that will not be expected by another layer, therefore we have to add $O(n-2\sqrt{n})$ message.

5.3.2. In the worst case

Lemma 4. *The algorithm shape shifting needs $3n-2\sqrt{n}+O(n)$ messages in the best case.*

The case that represents the worst case for the number of message exchanged is when at the beginning the shape is a straight chain because message transmission cannot be carried in parallel. Therefore in each layer $|layer|$ messages are to be added, matching exactly to $n-1$ messages.

Theorem 5. *From Lemma 1, Lemma 2, Lemma 3 and Lemma 4 the proposed protocol needs $3n$ of memory since the algorithm initiator election and shape shifting do not executed in parallel. And needs $4n-2\sqrt{n}+O(n)-1$ messages in the best case. And $5n-2\sqrt{n}+O(n)$ in the worst case.*

6. Simulation

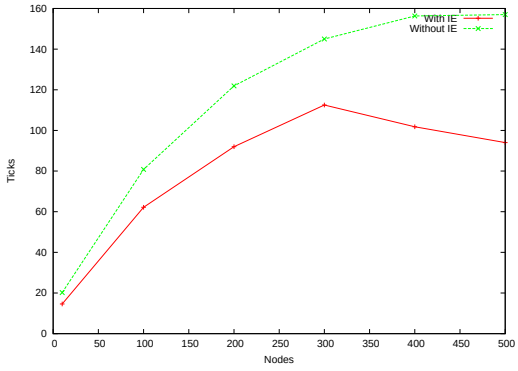

Figure 11: Execution time of the algorithm

We have done the simulations with the Dynamic Physical Rendering Simulator (DPRSim), which is a tool for use in the dynamic physical rendering project. It is a multithreaded platform on which we can develop and test new distributed algorithms for large ensembles of nodes; it represents a real environment of distributed algorithms for implementing distributed algorithm dedicated to MEMS microrobots. In our simulations, the radius of the node is 1 mm. The simulations were executed on a laptop with an Intel(R) Core(Tm) i5 at 2.53 GHz processor and 4 GB of memory. The following simulation results are the average of 100 tests (for each number of nodes) on the physical

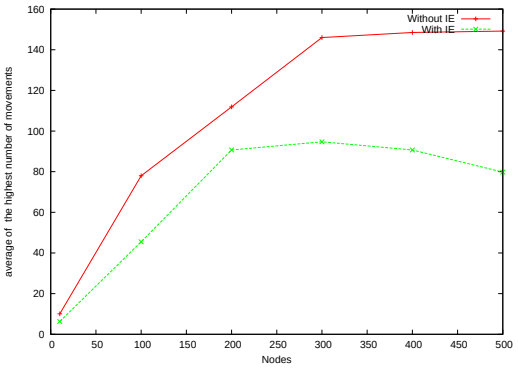

Figure 12: Average of the overall number of movements in the network based on the number of nodes

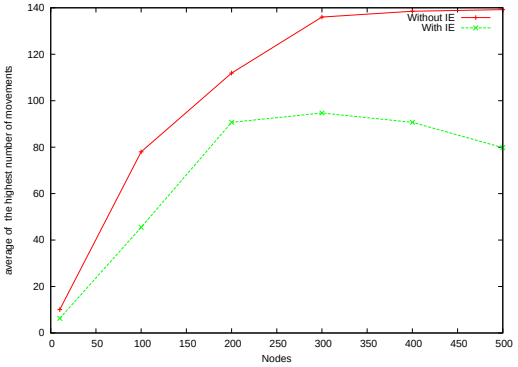

Figure 13: Average of the highest number of movements in the network based on the number of nodes

topology of connected and randomly generated networks of 10, 100, 200, 300, 400 and 500 nodes in 50 x 50 (mm^2) simulation area respectively. Figures 14, 15, 16, 17 and 18 represent instances of execution of the self-reconfiguration algorithm. The figures in this section compare the simulation results with initiator election algorithm (in figures of simulation, IE is an abbreviation of Initiator Election) and results with random choice of the initiator in the networks (without initiator election). We recall that the initiator election objective is to obtain a minimal number of movements, a good execution time, therefore an efficiency in energy consumption. The figures compare the execution time, the average of the number of movements (consumed energy), and the average of the highest number of movements.

Figure 14: Instance of execution: T1

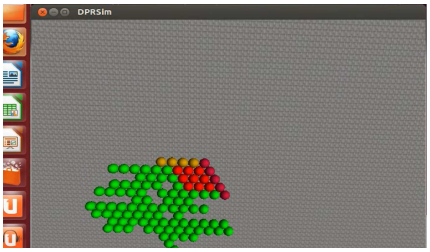

Figure 15: Instance of execution: T2

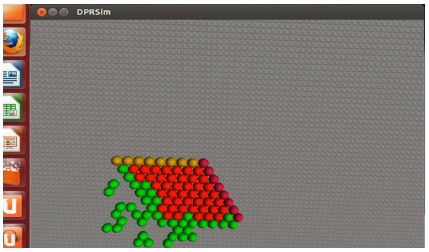

Figure 17: Instance of execution: T4

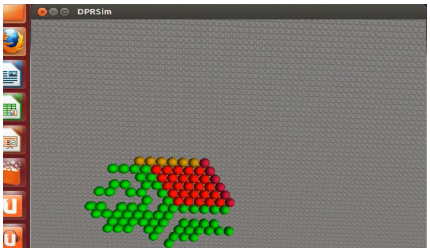

Figure 16: Instance of execution: T3

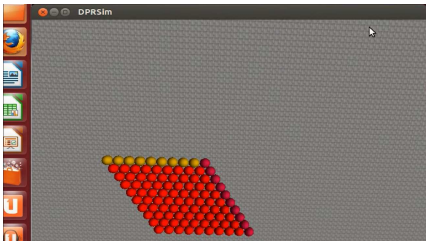

Figure 18: Instance of execution: T5

Figure 11 represents the execution time based on the number of nodes, Figure 12 represents the average of the overall number of movements in the network based on the number of nodes and Figure 13 represents the average of the highest number of movements in the network based on the number of nodes. In Figure 11 we see clearly the effect of the initiator election algorithm to optimize the execution time of the protocol, we see that whenever the network size increases the difference increases dramatically until 200 nodes.

An interesting thing to notice is that optimizing the execution time of the algorithm will have a direct effect on messages complexity, therefore a gain on communication, and if the algorithm is fast then the critical information arrives early at the concerned node. Also, if the task is a heavy parallel computation, therefore if the algorithm is faster, the parallel computing will be fast and light on the nodes because the tasks are well distributed. In figures 12 and 13 we see that whenever the network size increases the difference in number of movements increases dramatically, which will increases the probability of lifetime of nodes, therefore the probability that the node continues its task (its movements), this is also improving the energy consumption. We see in the results in figures that from 200 nodes, the curves decrease. This is because the number of nodes becomes large and nodes cover a large part of the simulation which gives a dense network. Therefore this change is the effect of the density of the network and the initiator election algorithm to have a good performance.

7. Conclusion

In this paper we proposed an improvement of the logical topology by using self-reconfiguration in MEMS microrobot networks. In this work, we proposed a protocol for self-reconfiguration of MEMS microrobots without map of the target shape (predefined positions of the target shape), which makes the algorithm efficient and scalable. The proposed solution deals with MEMS microrobots characteristics as the limited energy and memory capacities. In the presented work, nodes do not record any position which will make the protocol memory-efficient. The presented protocol is the first that provides a self-reconfiguration standalone and portable because it is independent of the map that builds the target shape starting from any connected network. The predefined positions of the target shape are replaced by the collaboration between nodes, which makes the algorithm more intelligent than a coordinate-based one.

However, some open problems remain; we will study the fault tolerance on self-reconfiguration in microrobot networks. The study of the effect of self-reconfiguration on the permutation routing [46] where the objective will be to optimize the path of a node to go to the correct position where it finds its correct data. Also, the use of tabu algorithms to achieve the self-reconfiguration for MEMS microrobots.

References

[1] T. Ebefors, J.U. Mattsson, E. K. lvesten, and G. Stemme, *A walking a silicon micro-robot*, in The 10th Int. Conf. on Solid-State Sensors and Actuators (Transducers '99), pages 1202-1205, Sendai, Japan, June 1999

[2] J. Mech. Design, *Microrobot design using fiber reinforced composites*, Journal of Mechanical Design vol. 130, no. 5, 2008

[3] B. Warneke, M. Last, B. Leibowitz, and K.S.J Pister,K.S.J., 2001, *Smart Dust: Communicating with a Cubic-Millimeter Computer*, Computer Magazine, pp. 44-51, 2001

[4] S. Hollar, A. Flynn, C. Bellew, and K.S.J. Pister, *Solar powered 10mg silicon robot*, In MEMS, Kyoto, Japan, January 2003

[5] J. Bourgeois and S.C. Goldstein. *Distributed Intelligent MEMS: Progresses and perspectives*, IEEE Systems Journal, 2014

[6] *http://today.duke.edu/2008/06/microrobots.html*

[7] *http://smartblocks.univ-fcomte.fr/*

[8] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. Pillai, *Meld: A Declarative Approach to Programming Ensembles,* In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '07), October, 2007

[9] M. P. Ashley-Rollman, P. Lee, S. C. Goldstein, Padmanabhan Pillai, and Jason D. Campbell, *A Language for Large Ensembles of Independently Executing Nodes,* In Proceedings of the International Conference on Logic Programming (ICLP '09), July, 2009

[10] S. Funiak, P. Pillai, M. P. Ashley-Rollman, J. D. Campbell, and S. C. Goldstein, *Distributed Localization of Modular Robot Ensembles,* In Proceedings of Robotics: Science and Systems, June 2008

[11] R. Ravichandran, G. Gordon, and S. C. Goldstein: *A Scalable Distributed Algorithm for Shape Transformation in Multi-Robot Systems,* In Proceedings of the IEEE International Conference on Intelligent Robots and Systems IROS '07, October, 2007

[12] M. E. Karagozler, A. Thaker, S. C. Goldstein, D. S. Ricketts, *Electrostatic Actuation and Control of Micro Robots Using a Post-Processed High-Voltage SOI CMOS Chip*, IEEE International Symposium on Circuits and Systems (ISCAS), May 2011

[13] E. Sahin. *Swarm robotics: from sources of inspiration to domains of application,* Swarm Robotics, SAB 2004 International Workshop (Revised Selected Papers) E. Sahin and W. M. Spear (Eds.), Lecture Notes in Computer Science 3342, Springer, 2005

[14] T. Balch and M. Hybinette. *Social potentials for scalable multi-robotformations,* In Proc. IEEE International Conference on Robotics and Automation, 2000

[15] S. Jeon, C. Ji, *Randomized Distributed Configuration Management of Wireless Networks: Multi-layer Markov Random Fields and Near-Optimality* CoRR abs/0809.1916, 2008

[16] K.Stoy, R.Nagpal, *Self-reconfiguration using Directed Growth*, 7th International Symposium on Distributed Autonomous Robotic Systems (DARs), France, June23-25, 2004

[17] E. Ferrante, A.E Turgut, C. Huepe, A. Stranieri, C. Pinciroli and M. Dorig, *Self-organized flocking with a mobile robot swarm: a novel motion control method*, in journal of Adaptive Behaviour, V.20, N.6, P.460-477, 2012

[18] M. Mamei, A. Roli, F. Zambonelli, *Emergence and Control of Macro Spatial Structures in Perturbed Cellular Automata, and Implications for Pervasive Computing Systems*, IEEE Transactions on Systems, Man, and Cybernetics, 36(5), May 2005

[19] M. Mamei, M. Vasirani, F. Zambonelli, *Experiments of Morphogenesis in Swarms of Simple Mobile Robots*, Journal of Applied Artificial Intelligence, 8(9-10):903-919, Oct. 2004

[20] F. Zambonelli, M.P. Gleizes, M. Mamei, R. Tolksdorf, *Spray Computers: Explorations in Self-Organization*, Journal of Pervasive and Mobile Computing, Elsevier, Vol. 1, p. 1-20, 2005

[21] F. Kribi, P. Minet, A. Laouiti, *Redeploying mobile wireless sensor networks with virtual forces*, IFIP Wireless Days, Paris, France, December 2009

[22] P. Minet, S. Mahfoudh, *Energy, bandwidth and time efficiency in data gathering applications*, IFIP Wireless Days, Paris, France, December 2009

[23] E. Milan, R. Tahiry and S. David, *Covering Points of Interest with Mobile Sensors*, IEEE Transaction on Parallel and Distributed Systems Vol.24, N.1, P.32-43, 2013

[24] K. Stoy, R. Nagpal, *Self-Repair Through Scale Independent Self-Reconfiguration,* Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robotsn and systems, Sendai, japan, 2004

[25] D. Rus, M. Vona, *Crystalline robots: Self-reconfiguration with compressible unit modules*, Autonomous Robots 10(1), 107-124, 2001

[26] P. White, V. Zykov, J. C. Bongard, H. Lipson, *Three dimensional stochastic reconfiguration of modular robots* In: Proceedings of Robotics Science and Systems, pp. 161-168. MIT Press, Cambridge, 2005

[27] Z. J. Butler, K. Kotay, D. Rus, K. Tomita, *Generic decentralized control for lattice-based self-reconfigurable robots*, International Journal of Robotics Research 23(9):919-937, 2004

[28] C. Jones, M. J. Mataric, *From local to global behavior in intelligent self-assembly.In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation,*ICRAM 2003, vol. 1, pp. 721-726. IEEE Computer Society Press, Los Alamitos, 2003

[29] W. Shen, P. Will and A. Galstyan, Hormone-inspired self-organization and distributed control of robotic swarms. Autonomous Robots 17(1), 93-105, 2004

[30] J. Bateau, A. Clark, K. McEachern, E. Schutze, and J. Walter, *Increasing the Efficiency of Distributed Goal-Filling Algorithms for Self-Reconfigurable Hexagonal Metamorphic Robots*, Proc. of the International Conference on Parallel and Distributed Techniques and Applications, PDPTA 2012, Las Vegas July 2012

[31] K. Stoy, *Using cellular automata and gradients to control self-reconfiguration*, Robotics and Autonomous Systems 54(2), 135-141, 2006

[32] H. Bojinov, A. Casal, T. Hogg, *Emergent structures in modular self-reconfigurable robots,* Proceedings of the IEEE International Conference on Robotics and Automation, vol. 2, pp. 1734-1741. IEEE Computer Society Press, Los Alamitos, 2000

[33] J. Walter, J. Welch, and N. Amato, *Distributed reconfiguration of metamorphic robot chains,* Springer-Verlag Journal on Distributed Computing, vol. 17, pp. 171-189, 2004

[34] J. Walter, B. Tsai, and N. Amato, *Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots,* IEEE Transactions on Robotics, vol. 21, no. 4, pp. 621-631, 2005

[35] S. Wong and J. Walter, *Deterministic Distributed Algorithm for Self-Reconfiguration of Modular Robots from Arbitrary to Straight Chain Configurations,*IEEE International Conference on Robotics and Automation, Germany, May 2013

[36] T. Larkworthy and S. Ramamoorthy, *An efficient algorithm for self-reconfiguration planning in a modular robot,* IEEE Intl. Conf. Robotics and Automation, pp. 5139-5146, 2010

[37] D. Dewey, S. S. Srinivasa, M. P. Ashley-Rollman, M. D. Rosa, P. Pillai, T. C. Mowry, J. D. Campbell, and S. C. Goldstein, *Generalizing Metamodules to Simplify Planning in Modular Robotic Systems,* In Proceedings of IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems IROS '08, September, 2008

[38] M. D. Rosa, S. C. Goldstein, P. Lee, J. D. Campbell, and P. Pillai, *Programming Modular Robots with Locally Distributed Predicates,* In Proceedings of the IEEE International Conference on Robotics and Automation ICRA, 2008

[39] H. Lakhlef, H. Mabed, J. Bourgeois, *Distributed and Efficient Algorithm for Self-reconfiguration of MEMS Microrobots*, in the 28th ACM Symposium On Applied Computing,p 560-566, Coimbra, Portugal, March 2013

[40] H. Lakhlef, H. Mabed, and J. Bourgeois, *Distributed and Dynamic Map-less Self-reconfiguration for Microrobot Networks.* In In IEEE NCA 2013, 12th IEEE International Symposium on Network Computing and Applications, Cambridge, MA, United States, pages 55–60, August 2013

[41] H. Lakhlef, H. Mabed, J. Bourgeois, *Parallel Self-reconfiguration for MEMS Microrobot*, in the 7-th IEEE International conference on Computer as a Tool, Zagreb, Croatia, pages 283–290, July 2013

[42] H. Lakhlef, H. Mabed, J. Bourgeois, *Dynamicity to Save Energy in Microrobots Reconfiguration*, in 10th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC-2013),Italy,pages 246–253, December 2013

[43] R. Moses, D. Krishnamurthy, R. Patterson. *A Self-Localization Method for Wireless Sensor Networks,*Eurasip Journal on Applied Signal Processing, 4:348-358, 2003

[44] A. Ward, A. Jones, A. Hopper, *A New Location Technique for the Active Office,* IEEE Personal Communications Magazine 4:42-7, 2002

[45] *The physical rendering simulator (dprsim)," http://www.pittsburgh intel-research.net/dprweb.*

[46] H. Lakhlef, J. F. Myoupo, *Secure permutation routing protocol in multi-hop wireess sensor networks,* International Conference on Security and Management (SAM'11), pp. 691-696, 2011