2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops

A Distributed Algorithm for a Reconfigurable Modular Surface

Didier El-Baz
CNRS - LAAS, 7 av. du colonel Roche F-31400 Toulouse
Université de Toulouse, LAAS, F-31400 Toulouse, France
Email : elbaz@laas.fr

Benoît Piranda and Julien Bourgeois
Université de Franche-Comté
- FEMTO-ST Institute, UMR CNRS 6174
1 Cours Leprince-Ringuet - 25200 Montbéliard, France
Email : {julien.bourgeois,benoit.piranda}@femto-st.fr

***Abstract*—A distributed algorithm is proposed in order to control block motion of a reconfigurable micro-electro-mechanical modular surface. The modular surface is designed to convey fragile and tiny micro-parts. The distributed algorithm solves a discrete trajectory optimization problem. In particular, the algorithm computes the shortest path between two points of the modular surface using a strategy based on minimum hop count. The proposed method based on distributed asynchronous iterative elections is scalable.**

***Keywords*-optimization, distributed computing, self-reconfigurable system, self-organizing system, MEMS, Smart Blocks, smart conveyor.**

I. Introduction

Most of the implemented solutions to sort and convey objects in production lines rely on contact-based technologies; this raises many questions. Fragile objects can be damaged or even scratched during manipulations. Medicines, food or micro-electronics parts can be contaminated (see [1]). This finally decreases the efficiency of the production line. Conveyors, based on air-jet technology, which avoid contact with conveyed parts tend to solve most of these problems (see [2]). Conveyors are generally designed as monolithic entities well suited to a specific task and fixed environment. As a consequence, conveyors have to be replaced if their environment changes; this occurs in particular if the input or output point of parts changes. New trends in robotics concern self-reconfigurable systems (see [3], [4], [5]). Some of these systems, which consist of small Micro-Electro-Mechanical Systems (MEMS) modules, can address dynamicity issue. In particular, they can bring flexibility in future productions lines. We note that MEMS-based devices with embedded intelligence, also referred to as distributed intelligent MEMS [6], [7] have great potentials on many fields and more particularly for manipulating micro parts in many industries like semiconductor industry and micromechanics (see [8], [9]).

Among a limited number of projects related to distributed reconfigurable smart conveyors, the Smart Blocks project [10] aims at designing a self-reconfigurable MEMS-based modular surface for safe and fast conveying of fragile micro parts. The Smart Blocks project aims at tackling all related problems so as to increase the efficiency of future production lines. The advantages of smart block conveyors are multiple: they can adapt easily to tasks changes and require less modules than a classic monolithic surface. The reader is referred to [11] for a complete and detailed presentation of the Smart Blocks project. The Smart Block project is a sequel to the Smart Surface project (see [12] and [13]). The Smart Surface project dealt with a MEMS-based monolithic conveyor that consisted of a distributed array of sensors and air-jet actuators.

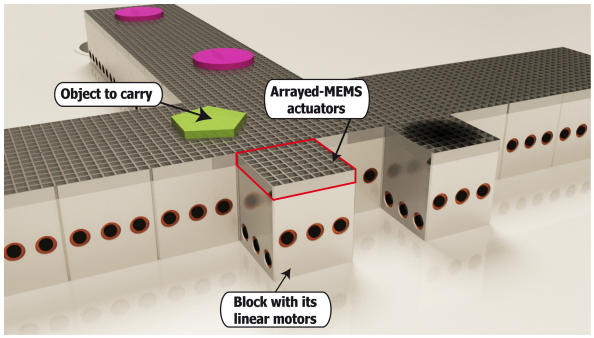

Fig. 1. The Smart Blocks modular conveyor.

In this paper, we make a very brief presentation of aspects related to robotics and technology. We concentrate on the design of a scalable distributed iterative algorithm that is well suited to shortest path problems. The algorithm deals with the solution of a discrete trajectory optimization problem. It is based on distributed election.

Due to technology constraints, the context considered in this paper is far more complex than the one considered in [14] since block motion necessitates here the presence of some other blocks, while blocks could move freely on the surface without any support of other blocks in our previous work.

Section II deals with technical aspects related to the modular surface in the Smart Block project. The model of the modular surface is presented in Section III. Section IV deals with block motion. The distributed algorithm is presented in section V. Section VI deals with conclusions and future work.

II. The modular surface

The centimeter scale modular surface studied in the Smart Block project is composed of few dozens of blocks.

A 2D pneumatic MEMS actuator array is embedded on the top of each block in order to move parts (see [2] and [11]).

978-1-4799-4116-2/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPSW.2014.178

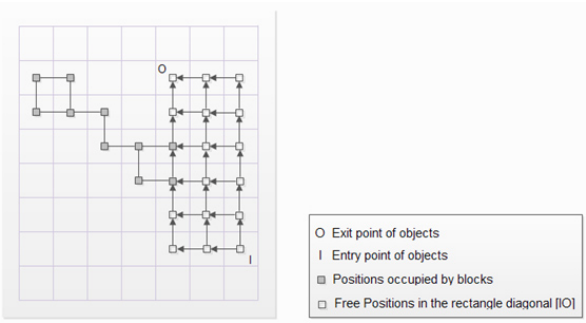

Fig. 2. Model of the modular surface.

Electro-permanent magnet-based actuators for block motion and sensors are also embedded on each side of a block (see Fig. 1). These features are used to detect neighboring blocks and to move blocks accordingly. Finally processing unit and communications ports are embedded in each block. As a consequence, block motion relies on contacts with other blocks and these contacts can occur only on each lateral side of a block, not on the top, nor the bottom of the block. The technology studied here is different from the one considered in [14] where only contact with the surface at the bottom of blocks was considered and block motion was not dependent on contact with other blocks except in the case where some block on the surface was an obstacle.

The context considered in this paper is thus far more constrained than the one considered in [14], since block motion necessitates here the presence of some blocks for support while blocks could move freely on the surface without any support of other blocks in our previous study. As a consequence, the strategies for block motion proposed in this paper are more complex than in [14].

III. The discrete model

We consider a discrete representation of the modular surface with two dimensional grid topology (see Fig. 2) where each node of the grid corresponds to the center of the cell that can be occupied by a block. In particular, small grey squares represent blocks. The input and output of parts are denoted by I and O, respectively. We consider the rectangle bounded by I and O, and denote by B_r the union of all nodes contained in this rectangle; we denote by L the set of links between the elements of B_r that are oriented from the input I to the output O. We obtain the oriented graph $G = (B_r, L)$ that is always oriented from the input to the output. For example, we have a left-up oriented graph if the output O is at left and above the input I as in Fig. 2. We note that all shortest paths between I and O are contained in the graph G. Small white squares represent empty cells (free nodes) that can be occupied by blocks.

The position of a node B on the surface is given by a two dimensional vector. The first component of this vector denoted by B_1 is an integer such that $0 \leq B_1 < W$, where W is the maximum width of the surface. The second component B_2 is an integer such that $0 \leq B_2 < H$, where H is the maximum height of the surface.
The components of I and O are denoted by $I_i, i \in \{1, 2\}$, and $O_i, i \in \{1, 2\}$, respectively.

The problem to solve is a discrete trajectory optimization problem between I and O. In order to solve this problem, we consider two metrics: the number of blocks along the shortest path between I and O and the number of hops that blocks must perform to build the shortest path.

Then, the problem consists in determining the strategy that minimizes block moves and that gives a shortest path between I and O in the oriented graph G. An optimal solution will minimize the number of blocks necessary to build the path between I and O, i.e., it corresponds to a shortest path with minimum hop count so that parts can be conveyed in minimum time. An optimal solution will also minimize block move, as a consequence, it minimizes the time needed to build the shortest path in order to satisfy industrial constraints.

We note that the maximum length of a shortest path on the surface is given by: $W + H - 1$; this value corresponds for example to the case where I and O are at the bottom right and top left corners of the surface, respectively.

IV. Block motion

Only straight moves, i.e., rectilinear block moves are allowed on the surface. Motions are limited by technological constraints, i.e., the use of electro-permanent magnet-based actuators. As a matter of fact, a block can move only if it is in contact with adjacent blocks. Generally speaking, a block motion relies on the support of adjacent blocks.

Managing global motion of a set of blocks via a distributed algorithm is not straightforward since it relies on the cooperation of several entities and combination of many consecutive elementary moves.

First, we study elementary block moves allowed by the physical system. Then, we propose to encode the deduced motion rules as an XML file. These rules simplify the validation process of all possible motions.

We consider the configuration of a set of blocks positioned over a 2D grid. An elementary motion moves a block to a neighboring position, i.e., to an adjacent cell. Nevertheless, we note that such an an elementary motion depends greatly on the state of adjacent cells.

In order to check if a block can move, we examine the initial state of the cell and neighboring cells, i.e., if their positions are initially occupied by blocks or not and the associated events that must be performed. To this end, we use a number to encode the state of a cell, i.e., 1 if the position is occupied by a block and 0 otherwise. Similarly, events at a given position, are encoded as shown in Table I, where six cases are possible: the context of the cell remains static, i.e., cell is empty or occupied by the same block, which correspond to codes 0 and 1, respectively; the context the cell is dynamic, i.e., an empty cell becomes occupied by a block, an occupied cell becomes empty or a block leaves a given cell that is occupied immediately by an adjacent block, which correspond to codes

TABLE I
CODES ASSOCIATED TO THE DIFFERENT EVENTS.

Code	Context	Case
0	Static	The cell remains empty
1	Static	The cell remains occupied by same block
2	Stat. or Dyn.	Every possible event can occur at that position
3	Dynamic	An empty cell becomes occupied
4	Dynamic	An occupied cell becomes empty
5	Dynamic	A new block occupies immediately a cell abandoned by a previous block

TABLE II
TRUTH TABLE FOR VALIDATION BLOCK MOTION.

Presence Matrix \ Motion Matrix	0	1	2	3	4	5
0	1	0	1	1	0	0
1	0	1	1	0	1	1

3, 4 and 5 respectively. Finally, the case where a cell does not have any incidence on a given motion is encoded by 2.

We introduce two types of local square matrices, i.e., the Motion and Presence Matrices, respectively. The Motion Matrix is associated to events related to a given block motion rule. The Presence Matrix is associated to the initial state of cells before the considered motion. More precisely, a Presence Matrix shows the state of a cell occupied by a block that is supposed to move (central entry of the square matrix) and the states of adjacent cells. For facility of presentation, we consider only 3x3 matrices in the sequel. In the general case, the size of the Presence Matrix and Motion Matrix can nevertheless be larger in order to take into account the simultaneous motion of set of blocks.

We concentrate first on a basic block motion and associated Motion Matrix. This motion corresponds to the case where a block moves to the right direction, sliding over two other blocks; it is called "east sliding" rule and the associated Motion Matrix is defined as follows.

$$M_M = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 4 & 3 \\ 2 & 1 & 1 \end{bmatrix}. \quad (1)$$

We consider the following 3x3 Presence Matrix.

$$M_P = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}. \quad (2)$$

We define the operator $M_M \bigotimes M_P$. The operator $M_M \bigotimes M_P$ applies Truth Table II to the corresponding entries of the matrices M_M and M_P (see Figure 3). Motion is valid if the result of the application of the Truth Table II is true for all entries, i.e., the resulting 3x3 matrix is filled by 1. In the particular case of the example quoted above, we obtain:

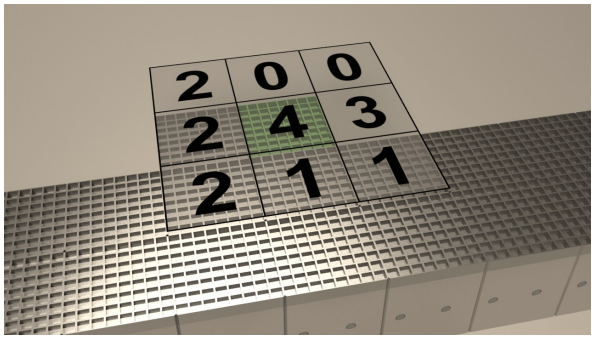

Fig. 3. "East sliding" rule: overlapping Motion Matrix and Presence Matrix.

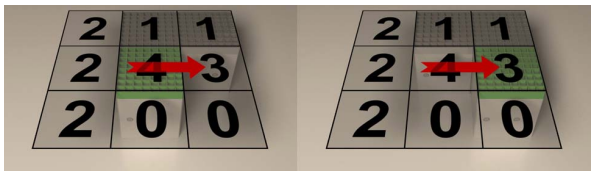

Fig. 4. "East sliding" rule: vertical symmetry.

$$\begin{bmatrix} 2 & 0 & 0 \\ 2 & 4 & 3 \\ 2 & 1 & 1 \end{bmatrix} \bigotimes \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \quad (3)$$

This confirms that the corresponding motion is valid. This rule allows the motion of a block from the central position (value 4) to the east position (value 3) if it exists two support blocks in the south of initial and final position of the moving block and free positions in the north.

Various block motion rules associated with different Motion Matrices can be introduced in order to represent several types of block motions that satisfy technology constraints. First of all, block motions can be derived via symmetry or rotation of a selected block motion, e.g., see Figure 4 for vertical symmetry. We note that there are situations where a selected block motion is not valid (see Figure 5).

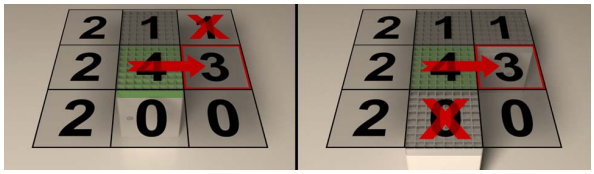

Fig. 5. Examples of situations where a given block motion is not valid.

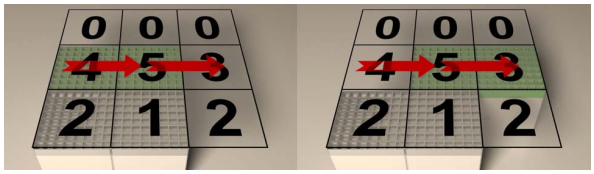

Fig. 6. "East carrying" rule.

```
<?xml version="1.0" encoding="utf-8"?>
<capabilities>
 <capability name="east1" size="3,3">
  <states>
   2 0 0
   2 4 3
   2 1 1
  </states>
  <motions>
   <motion time="0" from="1,1" to="2,1"/>
  </motions>
 </capability>
 <capability name="carry_east1" size="3,3">
  <states>
   0 0 0
   4 5 3
   2 1 2
  </states>
  <motions>
   <motion time="0" from="1,1" to="2,1"/>
   <motion time="0" from="0,1" to="1,1"/>
  </motions>
 </capability>
</capabilities>
```

Fig. 7. Extract of the XML code for block motion description

An important family of block motions corresponds to the case where several adjacent blocks move simultaneously, e.g., adjacent blocks in the same row or in the same column. As an example, the so-called "east carrying" rule corresponds to the following Motion Matrix (see Figure 6):

$$M_M = \begin{bmatrix} 0 & 0 & 0 \\ 4 & 5 & 3 \\ 2 & 1 & 2 \end{bmatrix} \tag{4}$$

This type of block motion is valid with the following Presence Matrix:

$$M_P = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{5}$$

Similar block motion rules can also be obtained via symmetry or rotation of Motion Matrix given in equation (4).

We encode block motion rules in an XML file that uses a simple vocabulary. For each rule, we give the Motion Matrix and the list of elementary moves. Fig. 7 shows an extract of the XML code that details the encoding of the two block motion rules presented in this section. For facility of presentation, we do not present here all the block motions rules and associated Motion Matrices that are allowed by technology constraints. Nevertheless, we note that a block motion that is not valid for a given Motion Matrix and Presence Matrix may be valid for the same Presence Matrix and a different Motion Matrix.

Without loss of generality, we can make the following assumption.

Assumption 1: All blocks are initially connected and the initial set of connected blocks has a two dimensional topology. As a consequence, initial patterns that consist only of horizontal or vertical series of blocks are excluded.

Remark 1: Block motion rules that disconnect one or several blocks are prohibited since a block that is separated from the set of blocks cannot move anymore due to technology constraints and thus cannot participate anymore to the distributed application. Block motion rules that could build a line or column of blocks between the input I and output O are prohibited. since this leads to a blocking.

V. Distributed algorithm

A. Principle of the distributed algorithm

The proposed approach presents the advantage to quickly set up a modular conveyor with minimum distance between an input, I, and an output, O, in compliance with industrial requirements. The proposed distributed algorithm relies on distributed MEMS computing paradigms (see [12], [13] and [15]).

Two discrete optimization problems are solved simultaneously by the proposed distributed iterative algorithm: a shortest path problem between two points of the modular surface (the input and output of parts) and the associated optimal moving of blocks necessary to build the shortest path subject to technology constraints.

Our algorithm is based on distributed iterative elections. At each iteration, a block is elected in an asynchronous distributed manner in order to move towards the output O. The election mechanism selects a block that is not in the same column or row as O and whose number of hops to reach a given position, i.e. the output, O, is minimal. Nevertheless, due to technology constraints regarding block motion presented in Section IV and contrarily to [14], the elected block does not move directly to the output, O (unless it is at one hop of O): it moves only to an adjacent cell (one hop motion towards O). This move along horizontal or vertical axis tends to diminish the distance between the elected block and O.

Without loss of generality and for facility of presentation, we consider in the sequel the following condition that is slightly more restrictive than *Assumption 1*.

Assumption 2: Initially, a block that is also called the Root, occupies position I and all blocks store in registers their position on the surface as well as the position of the output O. The set of blocks is connected with a two-dimensional topology excluding situations where all blocks but the Root occupy the same line or column between input I and output O.

We make the following additional assumption.

Assumption 3: All communications between adjacent blocks occur in finite time.

Details of the distributed iterative algorithm are presented in Algorithm 1, where $P\left(B^k\right)$ denotes the position of block B^k on the surface.

Algorithm 1: Distributed iterative algorithm

k=0;
Distributed election of block B^k;
while $P(B^k) \neq O$ **do**
  k=k+1;
  Distributed election of block B^k;
  B^k performs one hop towards O;
end

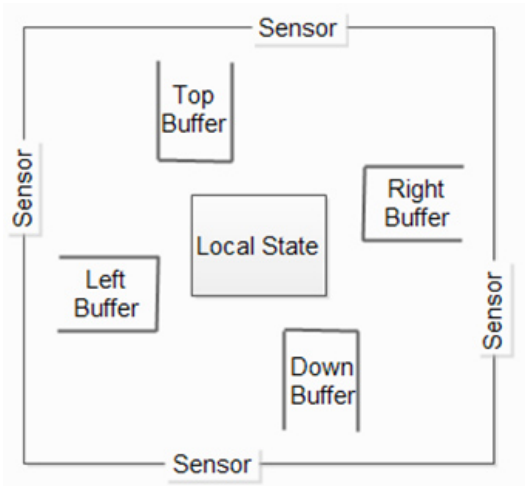

Fig. 8. Memory organization for data communication between blocks.

B. Memory organization of a block

The local state of a block is stored in a set of variables and tables. For example, the Neighbor Table, denoted by NT, stores information regarding blocks that may be connected to a given block. In the sequel, the iteration number is denoted by IT. Memory organization for data communication between blocks is displayed on Fig. 8. For a typical block with four neighbors, data sent by neighbors are stored in a dedicated buffer, e.g., top buffer, for neighbor that is above the considered block and right buffer for neighbor that is situated on the right side of the block (see Figures 8 and 9).

C. Distributed election

The method used in this paper for distributed election is based on the distributed procedure of Dijkstra and Scholten (see [16], see also [17]). The procedure is based on activity graph and acknowledgment of messages. In the beginning, only the block situated at Input I, called the Root is active. The Root starts computation, i.e., distributed election by sending an activation message to its neighbors. Each activation message activates a neighboring block that becomes a Son of the Root. The Root is also called the $Father$. Typically, activation messages are of the type:

$$Activate\,[Father, Son, O, ShortestDistance, IDshortest]$$

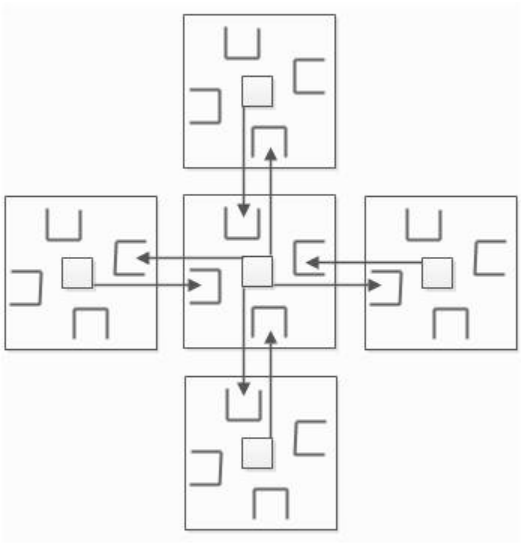

Fig. 9. Block communication scheme.

where the different fields of the message are: the ID of the sender ($Father$), the ID of the destination (Son), the location of the Output O, the current shortest recorded distance from a block to the output O, and the ID of the block with shortest recorded distance to O, respectively. Initially, we have

$$ShortestDistance = |O_i - I_i| + |O_j - I_j|\,, \tag{6}$$

and

$$IDshortest = Father. \tag{7}$$

Upon reception of an activation message, a son computes its distance to the output O. The distance of a block B to O is given by:

$$d_{BO} = +\infty, \text{if } B_i = O_i \text{ or } B_j = O_j, \tag{8}$$

$$d_{BO} = +\infty, \text{ if no move is possible for } B, \tag{9}$$

$$d_{BO} = |O_i - B_i| + |O_j - B_j|\,, \text{ otherwise.} \tag{10}$$

Equation (8) traduces the fact that the path between I and O must be as straight as possible. As a consequence, if I and O are on the same row of column and a block has already joined a position on this row or column, then this position must continue to be occupied by a block till the end of the distributed iterative process.

If d_{BO} is smaller than $ShortestDistance$, then $ShortestDistance$ is updated and takes value d_{BO}.

As the computation progresses, the activity graph evolves and more and more blocks become active. At some finite time all blocks have been activated. If an active block receives an activation message from a neighbor, then it does nothing. Active blocks that cannot activate neighbors anymore since they dont have a neighbor, but their father, or since all their neighbors have been activated by other blocks become inactive and send an acknowledgment message to their father. Similarly, active blocks that have

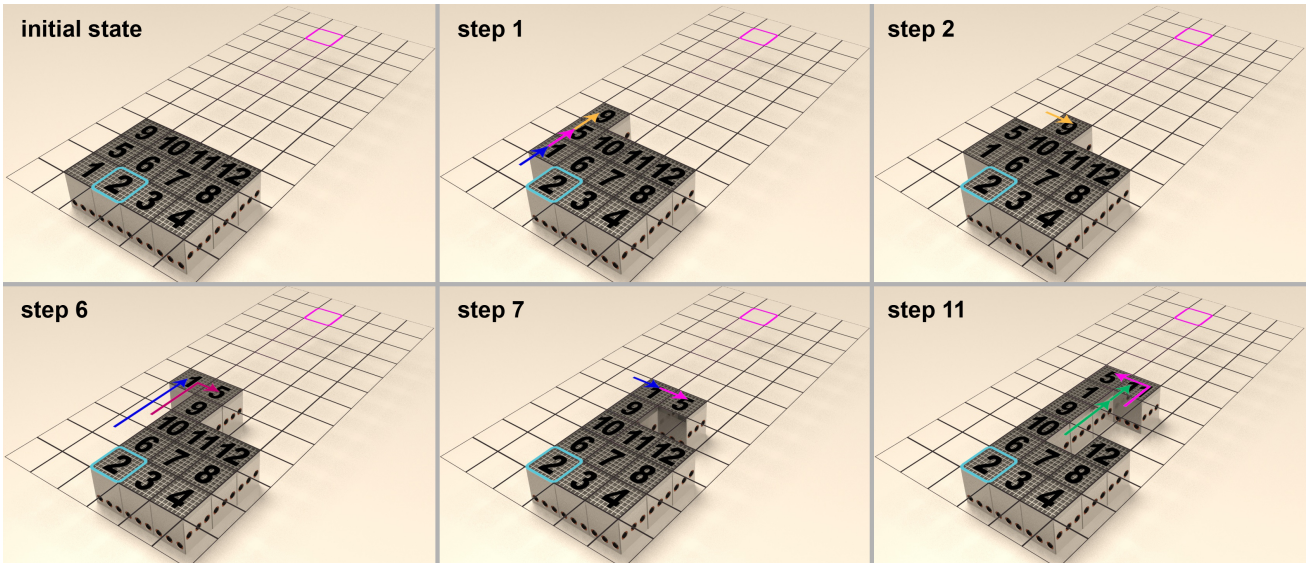

Fig. 10. Example of reconfiguration steps, beginning of the reconfiguration.

received acknowledgments from all their sons become inactive and send an acknowledgment message to their father. Acknowledgment messages are of the type:

$$Ack\left[Son, Father, ShortestDistance, IDshortest\right],$$

where the different fields of the message are, the ID of the sender (Son), the ID of the destination ($Father$), the current shortest recorded distance from a block to the output O, and the ID of the block with shortest recorded distance to O, respectively.

In the end, only the Root is active. This ends the first phase of the election algorithm. The Root then selects the block with shortest distance to the output O. If there are several blocks with the same shortest distance to O, then the Root selects randomly one block and sends a Select message to the elected block. The selection message is routed to the elected block according to the father/son path obtained in the first phase of the election algorithm.

The Elected block sends an acknowledgment message to the Root. Upon reception of the acknowledgment message, the Root becomes inactive. The distributed election is then terminated. The elected block can thus make an horizontal or vertical hop in the direction of the output O, so that the number of hops to reach O will be less important from the new position of this block; this is made according to the rules presented in Sections IV and V. We note that the local state of a block during a distributed election is given by a variable father, a table of sons, a table of acknowledged Activation messages, a variable d_{BO} and variable $ShortestDistance$.

Lemma 1: Under Assumptions 2 and 3, any trajectory optimization problem between the input I and outpout O, with shortest path length $N-1$, can be solved in finite time with at most N blocks by the proposed distributed algorithm.

Proof: The proof is in six steps.

a) It follows from Assumption 2 that one block occupies initially the input cell I. Moreover, the set of blocks is initially connected with a two dimensional topology excluding situations where all blocks but the Root occupy the same line or column between input I and output O.

b) It follows from the definition of the Distributed Algorithm 1 that positions on the shortest path that are occupied by blocks remain occupied all along the distributed application; though the IDs of the occupying blocks may change.

c) It follows from Remark 1 that motion rules presented in Section IV maintain a connected two dimensional topology for the set of blocks that excludes situations where all blocks that are not yet on the shortest path occupy the same line or column between input I and output O.

d) Assume now that no block can move at a given step s, i.e., all modules are blocked, then

- either no available block has support from adjacent blocks or all blocks that are not yet on the shortest path occupy the same line or column between input I and output O, which is not possible according to c);
- or a block has reached output O according to the distributed algorithm presented in Section V.

e) It follows from b), d) and Assumption 3 that the shortest path is buildt in finite time.

f) Finally, only the construction of shortest paths with length $N-1$ can be guaranteed with N blocks due to motion rules.

Remark 2: The computation complexity of the algorithm, i.e., the number of distance computation, is: $O\left(N^3\right)$, where N denotes the number of blocks.

Remark 3: The communication complexity of the algorithm, i.e., the number of messages exchanged between blocks is: $O\left(N^3\right)$.

Remark 4: The maximum number of block hops necessary to build the shortest path is: $O\left(N^2\right)$.

D. Example

We present now an example that illustrates the global behavior of the distributed algorithm. This example corresponds to a case with twelve blocks and shortest path distance between I, and O, equal to eleven. We consider here a limited number of families of block motion rules that have been introducted in Section IV. The shortest path is obtained after 55 block moves. Fig. 10 and Fig. 11 display a simulation of the modular surface and gives main steps of the reconfiguration. Blocks are identified by a number in order to follow their progression.
In Fig. 10, the initial state is displayed, position of input I, that is occupied by block #2, is represented by the blue rounded square at the bottom left of the figure and position of the output O, in the same column, is drawned by a magenta rounded square at the top. The first two steps display examples of motions allowing block #9 to cross the corner of the set of blocks, and block #1 and #5 to follow the motion. Block #5 is essential to enable block #9 to cross the corner, it carries block #9 beyond block #10 in order to allow it to move to the right. Steps 6, 7 and 11 show how blocks contribute to build the column of blocks from I to O. Fig. 11 displays some of the last steps of the reconfiguration. We note that the block #2 does not belong to the shortest path from I to O but it is essential to the constructuction of such path.

E. Simulations

We have developed a software[1] called VisibleSim [18] in order to visualize and debug, in real-time, distributed programs executed in a 3D environment with intelligent objects that are able to sense and act. VisibleSim mixes a discrete-event core simulator with discrete-time functionalities in the most efficient way so that simulations can scale up in numbers. We reported simulations with 2 millions of nodes at a rate of 650k events/sec on a simple laptop.

VisibleSim is, therefore, used for assessing the states of the blocks during the reconfiguration since it allows the observation of the asynchronous execution of the code on the different blocks. There are two options for implementing an algorithm inside VisibleSim. The first one is to use the Meld language [19] running on top of a virtual machine. Meld is a declarative programming language more specifically, a logic programming language able to be compiled on distributed environments. The virtual machines are all linked to the simulation cores which orchestrate the execution. The second option is to develop directly the program inside VisibleSim. A program, called a BlockCode, can be associated to each block. As VisibleSim is written in C++, the BlockCode has to be written in C++ too in order to inherit of the properties of the Block class.

Given the nature of our algorithm, it was easier to implement it using C++ than Meld. VisibleSim has helped debugging the program by changing the color of the blocks during the program or by writing debugging text, to name a few. The positions of the blocks are displayed in real-time after each move. A block can access the list of possible motions that are stored in the XML code presented in Figure 7.

All the images presented in this article have been realized by an external rendering software from 3D scene exported from VisibleSim.

[1]VisibleSim is available for download at http://github.com/claytronics/visiblesim

VI. Conclusions

In this paper, we have proposed a distributed iterative algorithm that solves a discrete trajectory optimization problem which occurs on a MEMS-based reconfigurable modular conveyor. The centimeter scale modular surface is used to convey millimeter-scale fragile objects via MEMS devices called blocks. Blocks cooperate to optimally build the shortest path between the input and output of parts on the surface. Electro-permanent magnet-based actuators for block motion impose many constraints. The proposed distributed approach presents the advantage to be scalable. A distributed election is implemented in order to obtain the block that will make the next hop on the surface. The distributed election is based on activity graph and acknowledgment of messages. The distributed approach studied in this paper is particularly useful to areas like semiconductors manufacturing, micro-mechanics and pharmaceutical industry since it is characterized by reconfigurability, flexibility, scalability and optimality that are key issues in the development of future production lines.

In order to complete our study, the proposed distributed algorithm will be carried out on an experimental centimeter scale self-reconfigurable smart blocks modular conveyor. We plan also to deal with fault detection, e.g., block failures, and sensor failures.

Acknowledgment

Part of this study has been made possible with the support of ANR-2011-BS03-005 grant and several fundings of the Labex ACTION (contract ANR-11-LABX-01-01) and ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F).

References

[1] *The rules governing medicinal products in the European Union*. Eudralex, 2010, ch. Good manufacturing practice guidelines.
[2] S. Konishi and H. Fujita, "A conveyance system using air flow based on the concept of distributed micro motion systems," *Journal of Microelectromecanical Syst.*, vol. 3, pp. 54–58, 1994.
[3] B. Salemi, M. Moll, and W.-M. Shen, "Superbot: A deployable, multifunctional, and modular self-reconfigurable robotic system," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2006, pp. 3636–3641.

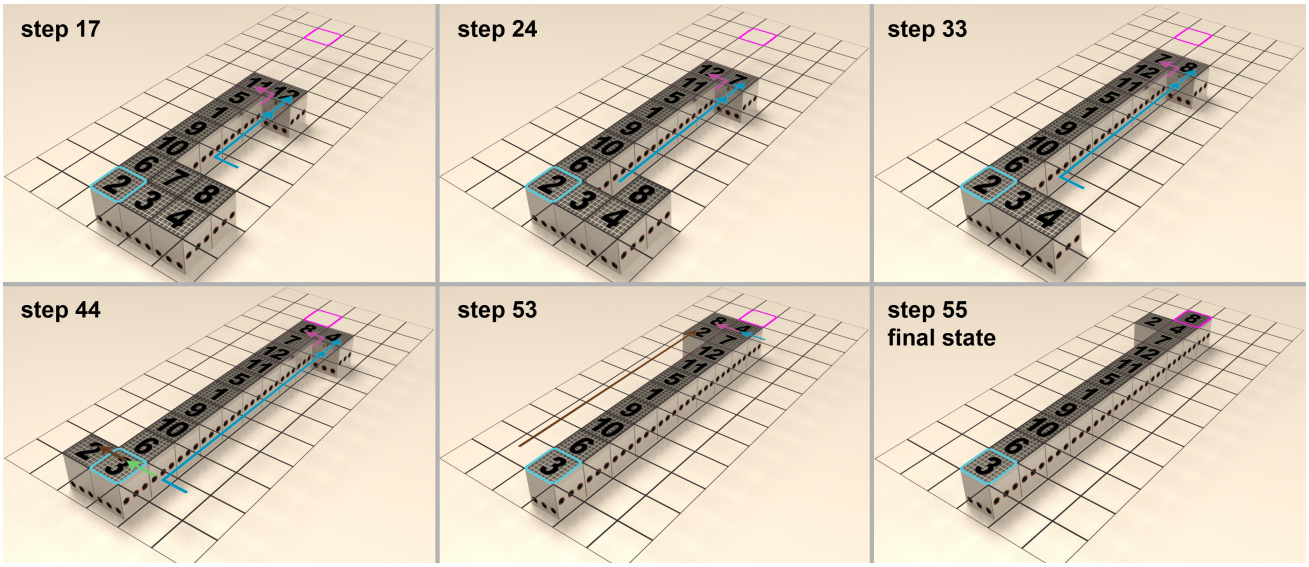

Fig. 11. Example of reconfiguration steps, end of the reconfiguration.

[4] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, Hasuo, and S. Murata, "Distributed self-reconfiguration of M-TRAN III modular robotic system," *Intl. J. Robotics Research*, vol. 27, pp. 373–386, 2008.

[5] V. Zykov, S. Mytilinaios, M. Desnoyer, and H. Lipson, "Evolved and designed self-reproducing modular robotics," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 308–319, 2007.

[6] J. Bourgeois and S. C. Goldstein, "Distributed intelligent MEMS: Progresses and perspectives," *IEEE Systems Journal*, 2013, accepted manuscript. To appear.

[7] J. Bourgeois and S. Goldstein, "Distributed intelligent MEMS: Progresses and perspectives," in *ICT Innovations 2011*, ser. Advances in Intelligent and Soft Computing, L. Kocarev, Ed. Springer Berlin / Heidelberg, 2012, vol. 150, pp. 15–25.

[8] D. Biegelsen, A. Berlin, P. Cheung, M. Fromherts, D. Goldberg, W. Jackson, B. Preas, J. Reich, and L. Swartz, "Airjet paper mover," in *SPIE Int. Symposium on Micromachining and Microfabrication*, 2000.

[9] Y. Fukuta, Y.-A. Chapuis, Y. Mita, and H. Fujita, "Design, fabrication and control of MEMS-based actuator arrays for air-flow distributed micromanipulation," *IEEE Journal of Micro-Electro-Mechanical Systems*, vol. 15, no. 4, pp. 912–926, Aug. 2006.

[10] J. Bourgeois. (2010, Jun.) smartblocks.univ-fcomte.fr. [Online]. Available: http://smartblocks.univ-fcomte.fr

[11] B. Piranda, G. J. Laurent, J. Bourgeois, C. Clévy, and N. Le Fort-Piat, "A new concept of planar self-reconfigurable modular robot for conveying microparts," *Mechatronics*, vol. 23, no. 7, pp. 906–915.

[12] K. Boutoustous, G. J. Laurent, E. Dedu, L. Matignon, J. Bourgeois, and N. L. Fort-Piat, "Distributed control architecture for smart surfaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan: IEEE, October 2010, pp. 2018–2024.

[13] D. El Baz, V. Boyer, J. Bourgeois, E. Dedu, and K. Boutoustous, "Distributed part differentiation in a smart surface," *Mechatronics*, vol. 22, no. 5, pp. 522–530, 2012.

[14] S. Tembo and D. El-Baz, "Distributed resolution of a trajectory optimization problem on a MEMS-based reconfigurable modular surface," in *2013 IEEE International Conference on Internet of Things (iThings/CPSCom)*. Beijing, China, August 2013.

[15] A. Berlin and K. Gabriel, "Distributed MEMS: New challenges for computation," *IEEE Computational Science and Engineering Journal*, vol. 4, no. 1, pp. 12–16, March 1997.

[16] E. W. Dijkstra and C.S.Scholten, "Termination detection for diffusing computations," *Inf. Proc. Letters*, vol. 11, no. 1, pp. 1–4, 1980.

[17] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[18] D. Dhoutaut, B. Piranda, and J. Bourgeois, "Efficient simulation of distributed sensing and control environment," in *IEEE International Conference on Internet of Things (iThings 2013)*, Beijing, China, Aug 2013, pp. 1–8.

[19] M. P. Ashley-Rollman, P. Lee, S. C. Goldstein, P. Pillai, and J. D. Campbell, "A language for large ensembles of independently executing nodes," in *Proceedings of the International Conference on Logic Programming (ICLP '09)*, July 2009.