A

A Distributed Algorithm for a Reconfigurable Modular Surface

Didier El-Baz, CNRS - LAAS
Benoît Piranda, UFC/FEMTO-ST Institute
Julien Bourgeois, UFC/FEMTO-ST Institute

A distributed algorithm is proposed in order to control block motion of a reconfigurable micro-electro-mechanical modular surface. The surface is designed to convey fragile and tiny micro-parts. The distributed algorithm solves a discrete trajectory optimization problem via swarm intelligence techniques. 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.

Categories and Subject Descriptors: []

Additional Key Words and Phrases: Swarm intelligence, distributed computing, self-reconfigurable algorithm, self-organizing method, MEMS, Smart Blocks, smart conveyor

ACM Reference Format:
Didier El-Baz, Benoît Piranda, Julien Bourgeois, 2014. A Distributed Algorithm for a Reconfigurable Modular Surface. *ACM Trans. Intell. Syst. Technol.* V, N, Article A (January 2014), 13 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. 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 [eud 2010]). 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 [Konishi and Fujita 1994]).
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 to convey changes. New trends in robotics concern self-reconfigurable systems (see [B. Salemi 2006; Kurokawa et al. 2008; Zykov V. and H. 2007]). 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 [Bourgeois and Goldstein 2013; 2012] have great potentials on many fields and more particularly for manipulating micro parts in many

This work has been funded by the Labex ACTION program (contract ANR-11-LABX-01-01) and ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F) and ANR (contract ANR-2011-BS03-005).
Author's addresses: D. El-Baz, CNRS - LAAS, Université de Toulouse, F-31400 Toulouse, France B. Piranda and J. Bourgeois, Université de Franche-Comté, FEMTO-ST Institute, UMR CNRS 6174, F-25200 Montbéliard, France.

DOI:http://dx.doi.org/10.1145/0000000.0000000

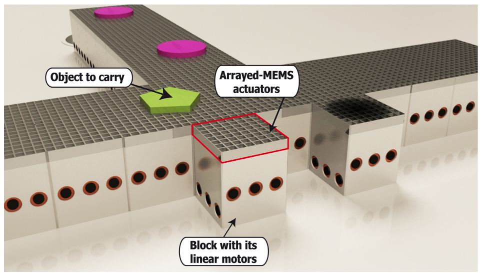

Fig. 1. The Smart Blocks modular conveyor.

industries like semiconductor industry and micromechanics (see [Biegelsen et al. 2000; Fukuta et al. 2006]).
Among a limited number of projects related to distributed reconfigurable smart conveyors, the Smart Blocks project [Bourgeois 2010] 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 [Piranda et al. 2013] for a complete and detailed presentation of the Smart Blocks project. The Smart Block project is a sequel to the Smart Surface project ([Boutoustous et al. 2010] and [El Baz et al. 2012]). The Smart Surface project dealt with a MEMS-based monolithic conveyor that consisted of a distributed array of sensors and air-jet actuators.
In this paper, we make a very brief presentation of aspects related to robotics and technology. We concentrate essentially 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 [Tembo and El-Baz 2013] since block motion necessitates here the presence of some other blocks, while blocks can move freely on the surface (without any support of other blocks) in [Tembo and El-Baz 2013]. As a consequence, the strategies for block motion proposed in this paper are more realistic than the one considered in [Tembo and El-Baz 2013].
Section 2 deals with technical aspects related to the modular surface in the Smart Block project. The model of the modular surface is presented in Section 3. The Section 4 deals with block motion. The distributed algorithm is presented in section 5. Section 6 deals with conclusions and future work.

2. 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 [Konishi and Fujita 1994] and [Piranda et al. 2013]). 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 essentially 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 totally different

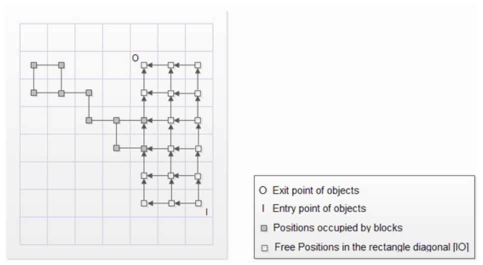
O
I
O Exit point of objects
I Entry point of objects
Positions occupied by blocks
Free Positions in the rectangle diagonal [IO]

Fig. 2. Model of the modular surface.

from the one considered in [Tembo and El-Baz 2013]. In [Tembo and El-Baz 2013], 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 [Tembo and El-Baz 2013], since block motion necessitates here the presence of some blocks for support while blocks can move freely on the surface (without any support of other blocks) in [Tembo and El-Baz 2013]. As a consequence, the strategies for block motion proposed in this paper are more complex than in [Tembo and El-Baz 2013].

3. THE DISCRETE MODEL

We consider a discrete representation of the modular surface with two dimensional grid topology (see Fig. 2). Each node of the grid corresponds to the center of the square 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 some 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 $I_{i,i}$ $\{1, 2\}$, and $O_{i,i}$ $\{1, 2\}$, 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 which minimizes block moves and that gives a final 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:

Table I. Cases and their codes

Code	Context	Case
0	Static	Position remains empty
1	Static	Position remains occupied by the same block
2	Static or Dynamic	Every possible event can occur at the position
3	Dynamic	Empty position becomes occupied by a block
4	Dynamic	Occupied position becomes empty
5	Dynamic	A new block occupies position abandoned by a block

Table II. Truth Table which permits one to validate block motion for a given occupation of cells.

	Validation Matrix					
Presence Matrix	0	1	2	3	4	5
0	1	0	1	1	0	0
1	0	1	1	0	1	1

$W + H - 1$; 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.

4. 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 using a distributed algorithm is not straightforward since it relies on the combination of many consecutive small 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 each possible motion.

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 implement a given motion, we compare the considered motion rule and the initiall state of its associated cell and its neighboring cells, i.e., if their positions are initially occupied or not by a block. 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, the cells that participate to a motion rule, i.e., the cell associated to a block and its neighboring cells 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 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 local 3x3 matrices, i.e., the Validation and Presence matrices, respectively that are associated to the motion rules and initial state of cells, respectively.

We concentrate first on a basic block motion and associated validation Matrix. This motion corresponds to the case where of a block moves to the right direction, sliding over two other blocks; it is called east sliding. We obtain the Validation Matrix of the equation (1):

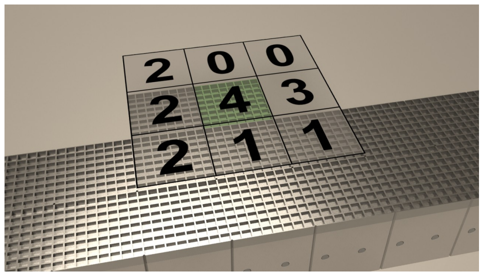

Fig. 3. Overlapping Validation Matrix and Presence Matrix.

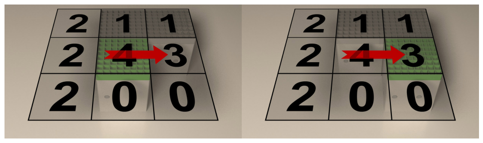

Fig. 4. East sliding rule (vertical symmetry).

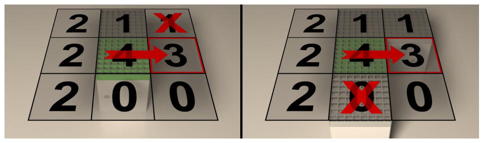

Fig. 5. Not allowed.

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

We consider the following 3x3 Presence Matrix.

$$M_P = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

We define the operator $M_V \bigotimes M_P$. The operator $M_V \bigotimes M_P$ applies Truth Table II to the corresponding entries of the matrices M_V and M_P (see Figure 3). Motion is possible 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 case of the above example, we obtain:

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

which confirms that the corresponding motion is valid. This rule authorizes 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 position in the north.
Similar block motion rules can be obtained via symmetry or rotations, e.g., see Figure 4 for vertical symmetry.

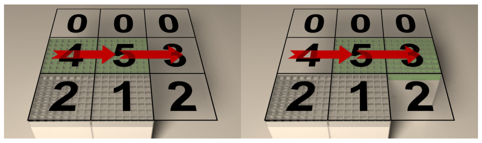

Fig. 6. East carrying rule.

From what was said previously, we deduce that some motions are not allowed under certain circumstances, e.g., see Figures 5.

Another family of block motion corresponds to the case where several adjacent blocks move simultaneously, e.g., adjacent blocks in the same raw or in the same column. As an example, the so-called east carrying rule corresponds to the following Validation Matrix (see Figure 6):

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

We note that motion is then possible 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 be obtained via symmetry or rotations.

We encode the several rules in an XML file that uses a simple vocabulary. For each rule, we have to give the validation matrix and the list of intermediary displacements. Fig. 7 shows an extract of XML code detailing the encoding of the two previous examples.

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.

5. DISTRIBUTED ALGORITHM

5.1. Principle of the distributed algorithm

The proposed approach presents the advantage to quickly set up a modular conveyor according to a shortest path between an input, I and an output, O; this is imposed by industrial requirements. The proposed distributed algorithm relies on distributed MEMS computing paradigms (see [Boutoustous et al. 2010] and [Berlin and Gabriel 1997]).

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

Our algorithm is mainly based on distributed iterative election. At each iteration, a block is elected in a distributed manner in order to move towards the output O. The election mechanism selects a block that is not in the same column or line as O

```
<?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 used for describing the rules

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
```

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 4 and contrarily to [Tembo and El-Baz 2013], the elected block does not move directly to the output, O (unless it is at one hop of O); the elected block makes only one hop 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 more restrictive than Assumption 1.

Assumption 2: The initial set of connected blocks has a two dimensional topology and a block of the connected set of blocks occupies initially position I. The input I and output O are initially known by the block situated at position I on the surface that is also called the Root. Finally, all blocks store in a register their initial position on the surface.

We make the following additional assumption.

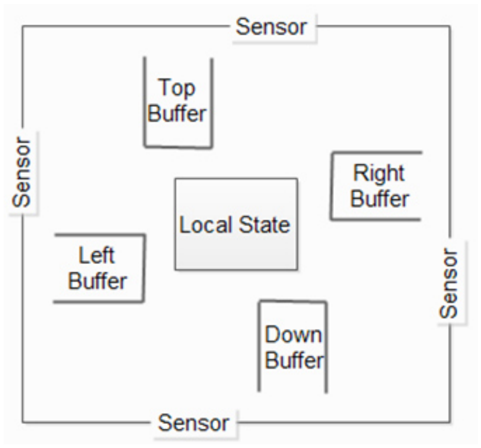

Fig. 8. Memory organization for data communication between blocks.

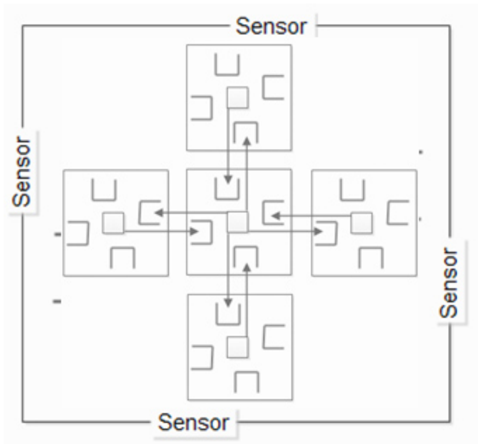

Fig. 9. Block communication scheme.

Assumption 3: All communications between adjacent blocks occur in finite time. Details of the distributed iterative algorithm are presented in Fig. 1, where $P\left(B^k\right)$ denotes the location of block B^k on the surface.

5.2. Memory organization of a block

The local state of a block on the surface is given by a set of variables and tables. For example, there is the iteration number, denoted by IT and the Neighbor Table, denoted by NT, which stores information regarding blocks that may be connected to the considered block. 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 Fig. 9).

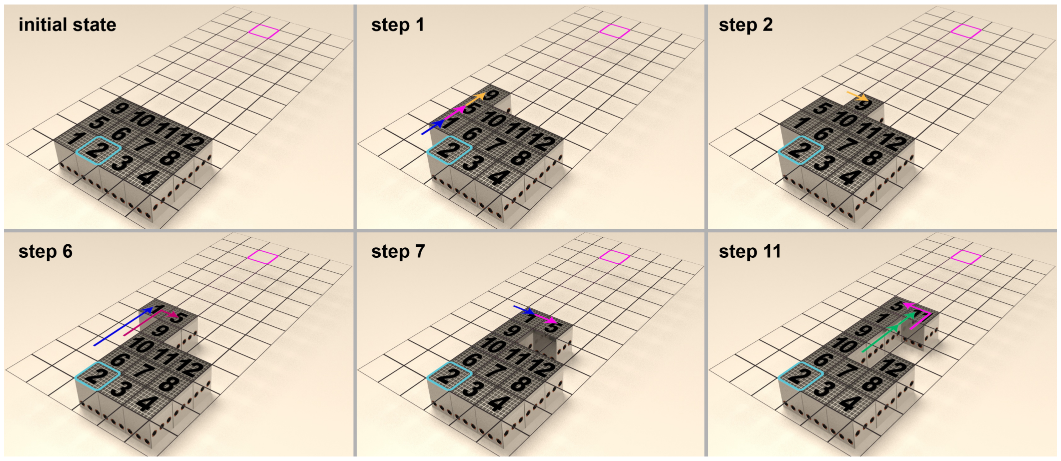

Fig. 10. Example of reconfiguration steps, beginning of the reconfiguration.

5.3. Distributed election

The method used in this paper for distributed election is based on the distributed procedure of Dijkstra and Scholten (see [Dijkstra and C.S.Scholten 1980], see also [Bertsekas and Tsitsiklis 1989]). 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:

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

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.

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:

$$d_{BO} = +\infty, \text{if } B_i = O_i \text{ or } B_j = O_j, \tag{6}$$

$$d_{BO} = +\infty, \text{ if no move is possible for } B, \tag{7}$$

$$d_{BO} = |O_i - B_i| + |O_j - B_j|\,, \text{ otherwise.} \tag{8}$$

Equation 6 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 line of column and a block has already join a position on this line 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 received acknowledgments from all their sons become inactive and send an acknowledgment message to their father. Acknowledgment messages are of the type: $Ack\,[Son, Father, ShortestDistance, IDshortest]$, 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 4 and 5. 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: If Assumptions 2 and 3 are valid, then any trajectory optimization problem with shortest path length N-1, can be solved in finite time with at most N blocks by our distributed algorithm.

Proof: Under Assumption 2 and motion rules presented in Sections 4 and 5, block motion is possible. Moreover, it follows from the distributed election mechanism presented above and Assumption 3 that some shortest path between an input and an output can be built in finite time. Nevertheless, due to motion rules, only the construction of shortest paths with length N-1 can be guaranteed with N blocks. See next sub-section for an illustration.

Remark 1: 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 2: The communication complexity of the algorithm, i.e., the number of messages exchanged between blocks is: $O\left(N^3\right)$.

Remark 3: The maximum number of block hops necessary to build the shortest path is: $O\left(N^2\right)$.

5.4. Example

We give 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. The shortest path is obtained after 55 block moves. Fig. 10 and Fig. 11 displays a simulation of the modular surface and gives main steps of the reconfiguration. Blocks are numbered in order to follow their progressions. In Fig. 10, the initial state is presented, I position is represented by the blue rounded square at the bottom left of the figure and O position is drawned by a magenta rounded

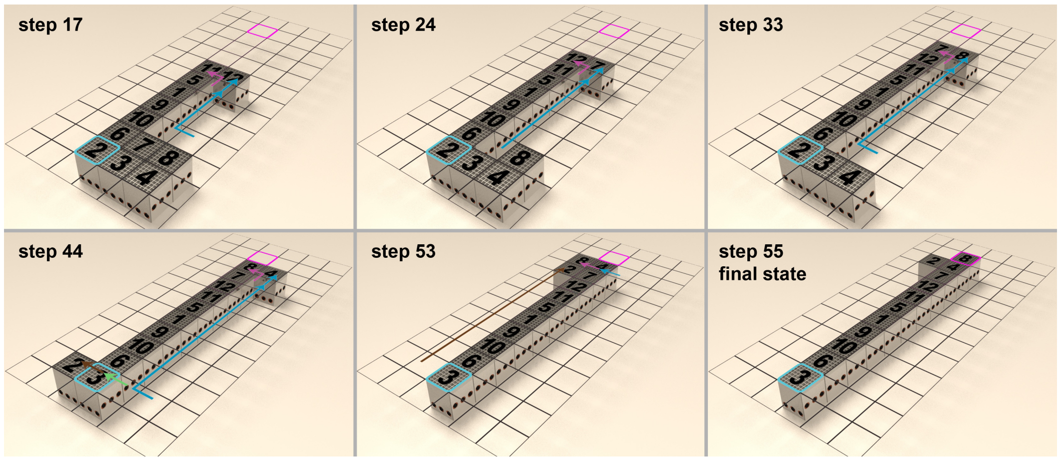

Fig. 11. Example of reconfiguration steps, end of the reconfiguration.

square at the top. The two first steps present an example of combinated motions allowing block#9 to cross the corner of the set of blocks, and block#1 to follow the same 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 present how blocks construct the line of blocks from I to O. Fig. 11 shows the last important steps of the reconfigurations. We can observe that the block#2 does not participate to the path from I to O but it is essential to the building of the path.
This example is fully presented in the joined video.

5.5. Simulation

We have developed a software[1] called VisibleSim [Dhoutaut et al. 2013] to visualize and to debug, in real-time, distributed programs executed in a 3D environment with intelligent objects 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 algorithm as 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 [Ashley-Rollman et al. 2009] 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 core which orchestrate the execution. The second one 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. It calculates the simultaneous evolution of the blocks state taking into account the message transmission delay. Thus, the emission of a message and its reception cannot be achieved simulta-

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

neously. After each movement, the positions of the blocks are displayed in real-time. Each block can access the list of possible movements 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.

6. CONCLUSION

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

REFERENCES

2010. *The rules governing medicinal products in the European Union*. Eudralex, Chapter Good manufacturing practice guidelines.

Michael P. Ashley-Rollman, Peter Lee, Seth Copen Goldstein, Padmanabhan Pillai, and Jason D. Campbell. 2009. A Language for Large Ensembles of Independently Executing Nodes. In *Proceedings of the International Conference on Logic Programming (ICLP '09)*.

W.-M. Shen B. Salemi, M. Moll. 2006. SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System. In *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*.

A. Berlin and Kaigham Gabriel. 1997. Distributed MEMS: New Challenges for Computation. *IEEE Computational Science and Engineering Journal* 4, 1 (March 1997), 12–16.

Dimitri P. Bertsekas and John N. Tsitsiklis. 1989. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

D. Biegelsen, A. Berlin, P. Cheung, M. Fromherts, D. Goldberg, W. Jackson, B. Preas, J. Reich, and L. Swartz. 2000. AirJet paper mover. In *SPIE Int. Symposium on Micromachining and Microfabrication*.

Julien Bourgeois. 2010. smartblocks.univ-fcomte.fr. (June 2010). http://smartblocks.univ-fcomte.fr

Julien Bourgeois and Seth Goldstein. 2012. Distributed Intelligent MEMS: Progresses and Perspectives. In *ICT Innovations 2011*, Ljupco Kocarev (Ed.). Advances in Intelligent and Soft Computing, Vol. 150. Springer Berlin / Heidelberg, 15–25.

Julien Bourgeois and Seth Copen Goldstein. 2013. Distributed Intelligent MEMS: Progresses and Perspectives. *IEEE Systems Journal* (2013). Accepted manuscript. To appear.

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

Dominique Dhoutaut, Benoit Piranda, and Julien Bourgeois. 2013. Efficient simulation of distributed sensing and control environment. In *IEEE International Conference on Internet of Things (iThings 2013)*. Beijing, China, 1–8.

Edsger W. Dijkstra and C.S.Scholten. 1980. Termination detection for diffusing computations. *Inf. Proc. Letters* 11, 1 (1980), 1–4.

D. El Baz, V. Boyer, J. Bourgeois, E. Dedu, and K. Boutoustous. 2012. Distributed part differentiation in a smart surface. *Mechatronics* (2012), 1–9.

Y. Fukuta, Y.-A. Chapuis, Y. Mita, and H. Fujita. 2006. Design, Fabrication and Control of MEMS-Based Actuator Arrays for Air-Flow Distributed Micromanipulation. *IEEE Journal of Micro-Electro-Mechanical Systems* 15, 4 (Aug. 2006), 912–926.

S. Konishi and H. Fujita. 1994. A conveyance system using air flow based on the concept of distributed micro motion systems. *Journal of Microelectromecanical Syst.* 3 (1994), 54–58.

H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, Hasuo, and S. Murata. 2008. Distributed Self-reconfiguration of M-TRAN III Modular Robotic System. *Intl. J. Robotics Research* 27 (2008), 373–386.

Benoit Piranda, Guillaume J. Laurent, Julien Bourgeois, Cédric Clévy, and Nadine Le Fort-Piat. 2013. A New Concept of Planar Self-Reconfigurable Modular Robot for Conveying Microparts. *Mechatronics* 23, 7 (Oct. 2013), 906–915. DOI:http://dx.doi.org/10.1016/j.mechatronics.2013.08.009

Serge Tembo and Didier El-Baz. 2013. Distributed Resolution of a Trajectory Optimization Problem on a MEMS-Based Reconfigurable Modular Surface. In *2013 IEEE and Internet of Things (iThings/CPSCom)*. Beijing, China.

Desnoyer M. Zykov V., Mytilinaios S. and Lipson H. 2007. Evolved and Designed Self-Reproducing Modular Robotics. *IEEE Transactions on Robotics* 23, 2 (2007), 308–319.