Distributed Resolution of a Trajectory Optimization Problem on a MEMS-based Reconfigurable Modular Surface

Serge Tembo, Didier El Baz

2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing

Distributed Resolution of a Trajectory Optimization Problem on a MEMS-based Reconfigurable Modular Surface

Serge Romaric Tembo, Didier El-Baz
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Université de Toulouse, LAAS, F-31400 Toulouse, France
e-mail : tembo.mouafo@laas.fr elbaz@laas.fr

***Abstract*— In this paper we propose a distributed algorithm to solve a discrete trajectory optimization problem that occurs in a micro-electro-mechanical based modular surface context. The method computes the shortest path between two points of the modular surface using a strategy based on minimum hop count. Our scalable approach is based on distributed asynchronous iterative elections. A multithreaded Java Smart Blocks Simulator used to validate our distributed algorithm is presented and some results obtained with the simulator are commented on.**

Keywords-component; MEMS; Smart blocks; Smart conveyor; distributed algorithm; self-reconfiguration.

I. INTRODUCTION

Several solutions have been proposed in order to sort and convey objects in production lines; most of them are contact-based technologies that can raise some problems. Fragile objects can be damaged or even scratched during manipulations which lower the production line efficiency. For example medicines [1], micro-electronics parts or even food can be contaminated. Conveyors which avoid contact with the parts conveyed solve most of these problems for transport and sorting.

Conveyors are usually designed as monolithic entities that are well suited to a specific problem with fixed type of environment. If for some external reason, the environment of the conveyor changes, e.g., a change of the usual input or output point of parts to convey, then the conveyor has to be reconfigured or even replaced. Self-reconfigurable systems [2-4] made of small Micro-Electro-Mechanical Systems (MEMS) modules can address this problem and can bring flexibility in future productions lines. In particular, MEMS-based devices with embedded intelligence, also referred to as Smart Blocks, have great potential for manipulating micro parts in many industries like semiconductor industry and micromechanics (see [5, 6]).

The Smart Blocks project [7] aims at designing a self-reconfigurable MEMS-based modular surface for fast conveying of fragile micro parts and medicinal products. The goal of this project is to tackle all related problems in order to increase the efficiency of future production lines. The reader is referred to [8] for a complete and detailed presentation of the Smart Blocks project. In this paper, we concentrate on the design of a scalable distributed algorithm that is well suited to the solution of a discrete trajectory optimization problem that occurs in a MEMS-based modular surface context where fragile micro parts have to be conveyed.

The modular surface is composed of blocks. A block embeds a MEMS actuator array [8], see also [9], in the upper face in order to move the parts, a motion actuator for block motion, a sensor on each of its four side to detect neighboring blocks, a processing unit and ports for communication with neighbors (see Fig. 1). Blocks move on the surface via electro-permanent magnet technology. Parts are moved on top of blocks via a two dimensional pneumatic actuator (see [8]).

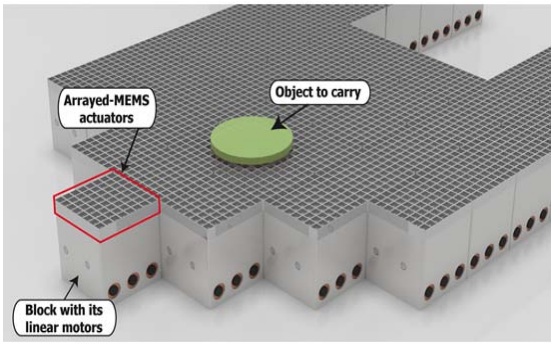


Figure 1. The Smart Blocks modular conveyor.

The proposed distributed algorithm deals with distributed MEMS computing paradigms [10, 11] and solves two discrete optimization problems simultaneously: the shortest path between two points of the modular surface (that will be used to convey parts) and the associated optimal moving of blocks necessary to build the shortest path. To achieve this task, we use a distributed iterative election method. At any iteration, a block whose number of hops to reach a given position on the surface is minimal is elected in a distributed manner and the elected block triggers the next iteration before it optimally moves to its final position. The distributed election is based on messages broadcasting with some control mechanisms to avoid loops and network flooding.

978-0-7695-5046-6/13 $26.00 © 2013 IEEE
DOI 10.1109/GreenCom-iThings-CPSCom.2013.128

Section II presents models of blocks and the modular surface. Section III deals with the principle of the distributed algorithm, while section IV details its design and the format of messages sent and received by blocks during the computation. Block motion is presented in section V. Section VI deals with SBS, a multi-threaded Java Smart Blocks Simulator which has permitted us to validate the proposed distributed algorithm. Section VII deals with simulation results and we conclude this study in section VIII.

II. MODELS OF BLOCKS AND MODULAR SURFACE

We consider a discrete representation of the MEMS-based modular surface with 2D grid topology. Each node of the grid corresponds to the center of the square that can be occupied by a block. The position of a node on the surface is given by a two dimensional vector. The first component x is an integer such that $0 \leq x < W$, where W is the maximum width of the surface. The second component y is an integer such that $0 \leq y < H$, where H is the maximum height of the surface (see Fig. 2). The parameters of the problem are the coordinates of the input and output of parts, that are denoted by I and O, respectively and the current position of blocks on 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 local state of a block on the surface is given by a set of parameters shown on Fig. 3. There are three ordinary flags denoted by TR (see section II for details), EL, CP (see section IV), respectively and an additional flag denoted by CE that can have three states (see section IV), a local iteration number denoted by IT, the Neighbor Table denoted by NT that stores information regarding blocks that may be connected to the considered block. The Waiting Message table denoted by WM stores the labels of message that are expected to be received from a neighbor. The QBEr table stores the labels of messages already forwarded by a block in order to avoid message loops and network flooding. We assume that the input I and output O are initially known by a block randomly chosen on the surface and denoted by B_i. This block sends a message, which is denoted by $QRY[ID_{srce}, IT, NULL, I_i, O_i]$, to all its neighbors and finally sets its TR flag (TR $= 1$) in order to avoid network flooding.

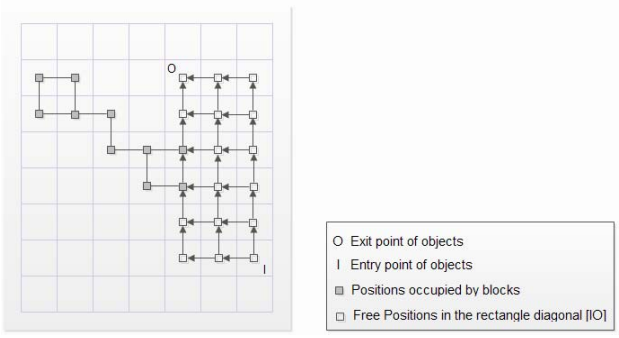


Figure 2. Model of the modular surface.

The first three fields of this message represent respectively the ID of the sender, the iteration number and the destination ID of the message (which is NULL in this case in order to denote multiple destinations). We also assume that a sensor on each side of a block, which detects the presence or departure of a neighboring block, permits one to update the Neighbor Table (denoted by NT). Each block which receives a QRY message, updates part its NT table and destroys the message if TR $= 1$; else the block sends the QRY message to all its neighbors, sets its TR flag and updates its NT table. Note that the sensors of a block and the embedded image of the block do not update the same field of the NT table. For example if the sensor of the upper side face detects a block, then it will set the field NT[Up]. $e = 1$ and if the ID of the block considered belongs to the rectangle bounded by I and O, then the embedded image will set the field $NT[Up].l$ to I(Incoming link) or O (Outgoing link) depending on the respective positions of I and O that give the orientation of the graph $G_0 = (B, L_0)$ where B is the set of blocks on the surface within the rectangle bounded by I and O and L_0 is the set of links between them. The graph is always oriented from the input to the output. For example, we shall have a Left-Up (LF-UP) oriented graph if the output O is at left and above the input I(see Fig. 2). Then broadcasting the QRY message to all blocks allows each block on the surface to receive the input parameters of the optimization problem and to take part to the construction of the graph G_0.

In order to solve the discrete trajectory optimization problem, we consider two metrics: the number of blocks along the shortest path and the number of hops blocks must perform to build the shortest path. The optimal solution is the path built with minimum number of blocks (shortest path with minimum hop count) in order to minimize the conveying time of parts and with minimal block motion in order to minimize the time needed to build the shortest path.

Now, we consider the rectangle R bounded by I and O, B_r the union of all free positions contained in R and all elements of B in R and L the set of links between the elements of B_r . We obtain a new oriented graph $G = (B_r, L)$ bounded by R (see Fig. 2). On Fig. 2, the smalls gray squares represent blocks and the smalls white squares represent some free positions on the surface which can be occupied by blocks.

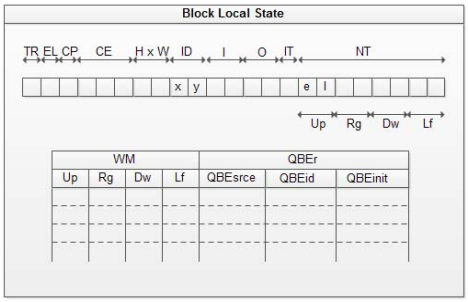


Figure 3. Local state of a block.

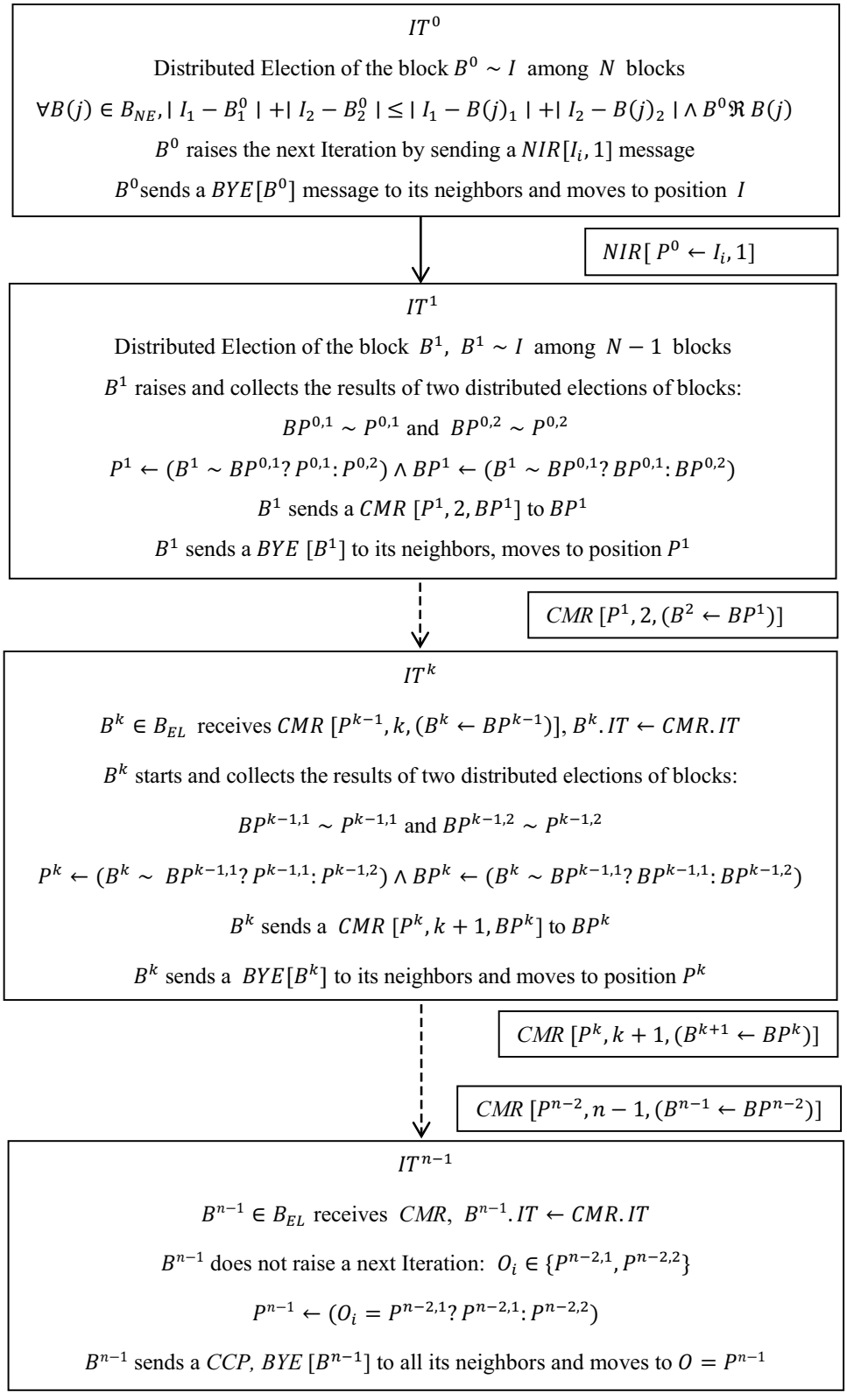


Figure 4. Distributed iterative algorithm.

We can easily note on Fig. 2 that all the paths between I and O, with minimal number of blocks, are contained in the oriented graph G. Then, the problem consists in determining the strategy which minimizes block moves and that gives a final shortest path in the oriented graph G.

In the proposed distributed approach, blocks cooperate to find out the shortest path while minimizing block moves at a global scale. Our approach is based on a distributed iterative election algorithm whereby at each iteration, the elected block is such that the number of hops to reach the next position on the shortest path is minimal.

III. PRINCIPLE OF THE DISTRIBUTED ALGORITHM

Without loss of generality, we assume that a connected set of N blocks is disposed initially on the surface and that there are enough blocks on the surface in order to build the shortest path between I and O. At iteration IT^0, the block whose number of hops to reach the entry point I is minimal, is elected among N blocks (see Fig. 4).

At iteration IT^k, where $1 < k < n-1$, the closest block to the position P^{k-1} on the shortest path (this block, denoted by B^k, has been elected among the available $N-k$ blocks at IT^{k-1}), has to choose either to move to the position $P^{k-1,1}$ or to the position $P^{k-1,2}$, where P^{k-1} is the final position on the shortest path of the block that has been elected at iteration IT^{k-2} and $P^{k-1,1}$, $P^{k-1,2}$ are the two outgoing positions from P^{k-1} in the graph G. The block B^k does not make this choice at random. The block B^k starts two distributed elections in order to obtain the position of blocks called $BP^{k-1,1}$ and $BP^{k-1,2}$, that are closest to the positions $P^{k-1,1}$ and $P^{k-1,2}$, respectively. Then, B^k moves either to the position $P^{k-1,1}$ or to the position $P^{k-1,2}$ depending on the proximity of $BP^{k-1,1}$ and $BP^{k-1,2}$ and the corresponding block will become B^{k+1}.

On Fig. 4, the relation $B(j) \sim P$ denotes that $B(j)$ is the closest block to the position P among the entire set of blocks on the surface. Moreover, the relation $P^k \leftarrow (B^k \sim BP^{k-1,1} ? P^{k-1,1} : P^{k-1,2})$ denotes that if B^k is the closest block to position of the block $BP^{k-1,1}$, then the value of P^k will be $P^{k-1,1}$, else it will be $P^{k-1,2}$.

Remark 1: Note that, in order to avoid any deadlock during an election due to the fact that it may exist more than one block at the same distance of a given position, we follow a strategy that favors a block that satisfies $B^k \,\Re\, B(j),\ j \in \{1, \dots, N-k\}$, where the relation $B^k \,\Re\, B(j)$ denotes: $B^k \sim O \vee B_1^k < B(j)_1 \vee B_2^k < B(j)_2, if\ B_1^k = B(j)_1 \forall B(j), j \in \{1, \dots, N-k\}$.

At iteration IT^{n-2}, where n is the maximum number of blocks necessary to build the optimal trajectory, the closest block to the output O is elected among $N-n+1$ blocks on the surface. This block will move to the output O at the last iteration IT^{n-1}.

Remark 2: Note also that the blocks already elected, i.e, with flag $EL = 1$, may participate to the next iterations by propagating messages.

Remark 3: At the beginning of the iteration $IT^k,\ 0 < k < n$, the set of N blocks on the surface is separated into two subsets. The first subset B_{EL} of k blocks already elected at previous iterations and a second subset of $N-k$ blocks not elected, denoted by B_{NE}.

IV. DISTRIBUTED ASYNCHRONOUS ITERATIVE ALGORITHM

In this section, we focus on the design of the distributed iterative election algorithm including transitions between consecutives iterations and message broadcasting mechanisms that are well suited to the grid topology of the modular surface.

A. Iteration IT^0

IT^0 is the first iteration of the distributed iterative algorithm. It performs the initialization of blocks and modular surface (see section II), it elects the block B^0 closest to the input I. Any block B participates to this election when it receives a $QRY[ID_{srce}, 0, NULL, I_i, O_i]$ message. The behavior of the block B depends on its context.

Case 1: Block B is closer to the input I than ID_{srce}.
If TR $= 0$, then the block B sends $QRY[B, 0, \text{NULL}, I_i,\ O_i]$ to all its neighbors, sets the flag TR to 1, updates the NT table in order to participate to the construction of the graph G, updates the (Waiting Message) WM table, i.e., it adds a label of a QRY message denoted by $QRY[ID_p, 0]$ into the WM$[p]$ table, $p \in \{\text{Up, Rg, Dw, Lf}\}$ and $p \neq p_{\text{srce}}$, where ID_p is the neighboring block located at the position p. The block B adds also the label of a message denoted by $BCI\ [ID_{srce}, 0, B]$ into WM$[p_{\text{srce}}]$ (which means that block B is waiting for a BCI message from ID_{srce} that will indicate if ID_{srce} has found out a block closer to the input I than B). The block B has a transition of its CE flag to the next state CE $= 01$ if CE $= 0$. Note that the transition order of the CE flag states is: $0 \rightarrow 01 \rightarrow 010$.
If $TR = 1$, then the Block B updates the WM table. It removes the label $QRY[ID_{srce}, 0]$ from the WM$[\,p_{srce}]$.

Case 2: Block B is farther to the input I than ID_{srce}.
If $TR = 0$, then the block B sends $QRY\ [B, 0, NULL, I_i,\ O_i]$ to all its neighbors, sets the flag TR to 1, updates the NT table in order to participate to the construction of the graph G, updates the WM table by adding a label $QRY[ID_p, 0]$ into the WM$[p]$ table with $p \in \{Up,\ Rg, Dw, Lf\}$ and $p \neq p_{srce}$. If block B does not have any neighbor other than the sender of the QRY message, then it must build and send a $BCI\ [B, 0, ID_{srce},\ ID_{srce}]$ message to the QRY sender, i.e., ID_{srce}. The last field of the BCI message contains the position of one block known by B as a block closest to the input I than ID_{srce}. If the block B has at least one neighbor other than the QRY sender, then it builds a message denoted

by $QBE[B, 0, NULL, Senders, ID_{srce}, I_i, O_i]$ and sends it to all these neighbors, adds a $RBE[ID_p, 0, ID_{srce}]$ label into each corresponding WM $[p]$ table, where $p \neq p_{srce}$. The field *Senders* is an ordered list of three IDs: the QBE initiator (B_i, in this case), the penultimate forwarder (NULL in this case since the block B is the initiator) and the last forwarder of the QBE message (NULL in this case) and ID_{srce} is here the so-called QBE_{ID} of the QBE message. Note also the input parameters I_i and O_i in the QBE message format in order to avoid deadlock if a block receives a QBE message without receiving and processing a QRY message. The block B sets its CE flag to the next state $CE = 010$ if $CE = 01$. Note that this last action will be performed only if the block B has already received and completed the processing of a QRY message from an outgoing position (if any), otherwise the CE flag state transition will be delayed until the block B has taken into account a QRY message coming from an outgoing position.

If $TR = 1$ then, block B behaves like in the previous case except that, it does not send a QRY message to its neighbor since it is already done.

The queries $QBE[QBE_{srce}, 0, QBE_{dest}, Senders, QBE_{ID}, I_i, O_i]$ eventually sent by blocks while processing of a QRY message are used to find out at least one block on the surface that is closest to the input I than the QRY sender (the so-called QBE_{ID}). Each block B on the surface which receives a QBE message first checks if it has not already treated this QBE. If it is not the case, i.e., the $QBEr$ table of the block B contains a QBE_{ID} entry, then block B deletes the QBE message and sends a Leave off Listening State message denoted by $LLS[B, 0, QBE_{srce}, RBE, QBE_{ID}]$ to the QBE sender i.e., QBE_{srce}, else block B has the following behavior.

Case 1: Block B is closest to the input I than QBE_{ID}.
Then block B builds and sends a message $RBE[B, 0, QBE_{srce}, B_i, QBE_{ID}, QBE_{init}]$ to the sender of the QBE message. The fourth field of this message is the answer denoted by RBE_{ANS} (B_i in this case) to the QBE request message received by the block. Note the QBE intiator ID (QBE_{init}) in the RBE message format.

Case 2: Block B is farther to the input I than QBE_{ID}.
If the length of the $Senders$ list is equal to 3, then the block B removes the second ID from the list and adds its own ID, i.e., B at the end of the list, else the block simply adds its ID. The block B forwards the request message $QBE[B, 0, NULL, Senders, QBE_{ID}, I_i, O_i]$ to its neighbors other than the QBE_{srce}, QBE_{ID}, QBE_{init}, the penultimate QBE message forwarder and any common neighboring block to one of these positions and B (this rule permits one to avoid unnecessary QBE and LLS messages processing). Block B adds a $RBE[ID_p, 0, QBE_{ID}]$ label into each corresponding WM$[p]$ where $p \neq p_{srce}$ and adds a QBE_{ID} entry into the $QBEr$ table.
If the block B does not have other neighbors, then it sends a $RBE\ [B, 0, QBE_{srce}, NULL, QBE_{ID}, QBE_{init}]$ to the QBE sender. The RBE_{ANS} is NULL here since the block B does not know a block closest to I than QBE_{ID}.

Case 3: Block B and QBE_{ID} are equidistant from the input I. The block B behaves like in the previous case except that transmission of a RBE message to the QBE sender has rather the ID of the block B, than a NULL RBE_{ANS} field.

A block B which has initiated or forwarded a QBE request message must collect all corresponding RBE response messages from its neighbors before to build its own RBE message or a BCI message if it is a QBE forwarder or a QBE initiator respectively. Note here the importance of the Leave off Listening State, $LLS[LLS_{srce}, 0, B_i, RBE, QBE_{ID}]$, to avoid deadlock. If the block B is waiting for a RBE message from the block LLS_{srce}, i.e., WM$[p_{srce}]$ table contains the $RBE[LLS_{srce}, 0, QBE_{ID}]$ label) and it receives a LLS message from the block LLS_{srce}, then the block B removes this label from the WM$[p_{srce}]$ table and stops waiting for RBE message from the neighbor LLS_{srce}.
If all RBE messages collected contains a NULL RBE_{ANS} field, then the block B sends a RBE containing also a NULL RBE_{ANS} field to the QBE sender or a BCI $[B, 0, QRY_{srce}, QRY_{srce}]$ message to the QRY sender, else the block B builts a RBE with the RBE_{ANS} field containing the closest ID to the input I among the RBE_{ANS} fields received or a BCI message with this closest ID as the value of the fourth field in the BCI message format. Note that the QRY sender block which receives a BCI message performs a transition the state of its CE flag according to the value of this fourth field. If this field contains the ID of the receiver block, then this block performs a transition of its CE flag state from the current state 0 to the next state 01, else CE flag state has a transition from 01 to 010.

A block is elected as being the closest to the input I when its current context is as follows: the flag TR has the value 1, the current state of the flag CE is 01 and the WM table is empty (it does not contain a message label). The block elected at this first iteration raises the second iteration by sending the (Next Iteration Request) $NIR[I_i, 1]$ message to one of its neighbors and starts moving to the input I.

Remark 4: It is not the end of an iteration which raises a new iteration, but the election of a block at the current iteration. This implies that there is no synchronization between the end of a iteration and the beginning of the next one. We thus have an asynchronous behavior. All the messages of a given iteration are obsolete at the next iteration. This explain why each block has a local iteration number and each message carries out an iteration number field (IT) which permits a block to know if it must destroy a received message (an obsolete message) or reset its local state and update its local iteration number before processing the message (if the local IT number is lower than the message IT number), or simply treat the message (if the local IT number is equal to the message IT number).

B. Iteration IT^1

This iteration is started by the block B^0 elected at iteration IT^0. The same processing as at iteration IT^0 is used in order to elect the block B^1 which is closest to the input I. The block B^1 must move to one of the two outgoing positions from I. In order to choose the optimal position, B^1 starts and collects the results of two distributed elections so as to elect the block closest to each outgoing position. The optimal outgoing position P^1 is the one with minimal hop count to one of the blocks obtained during the two elections; moreover, the corresponding block become the so-called block B^2. Then, B^1 sends a (Controlled Moving Request) $CMR\ [P^1, 2,\ B^2]$ message to the block B^2 and moves to the position P^1. The block B^2starts the second iteration when it receives the CMR message.

C. Iteration IT^k

The block B^k (elected at IT^{k-1}, $1 < k < n-1$) starts and collects the results of two distributed elections in order to choose the optimal outgoing position from P^{k-1} towards which the block B^{k-1} will move. The block B^{k+1} is obtained as a result of this process.

D. Iteration IT^{n-1} *(last iteration)*

The block B^{n-1} elected at IT^{n-2} sends a Come Closer to the Path (CCP) message to all its neighbors and moves to the output O. Each block not elected, i.e., with flag EL $= 0$ which receives a CCP message deletes this message if the flag $CP = 1$, else it propagates this message to its neighbors except the CCP sender, it sets its flag $CP = 1$ and moves to the position contained in the received CCP message. During the motion of a nonelected block, if the block encounters another block on its trajectory, then it waits until the block encountered has moved. This behavior at the last iteration permits one to have all blocks connected.

Remark 5: The complexity of the algorithm, i.e., the maximum number of block hops necessary to build the shortest path, is: $O(N.(W+H))$, where N denotes the number of blocks and W, H are the width and height of the surface, respectively.

V. BLOCK MOTION

When a block B is elected it must move from its current position P to its final position D. An actuator embedded in the block and controlled by the embedded image is responsible of block motion (see Section I). We detail in this section the algorithm run by the embedded image. We consider the rectangle bounded by the positions P and D. We note that all the optimal trajectories are contained in the oriented graph bounded by the considered rectangle.

At each hop, the block B updates its current position P and checks if a new hop is possible. Then, the block B moves to the next position. The behavior of a block can be modeled by a state machine whereby a block can have the following three possible states: moving, blocked (no hop is possible) and stopped (block is not moving), see Fig. 5.

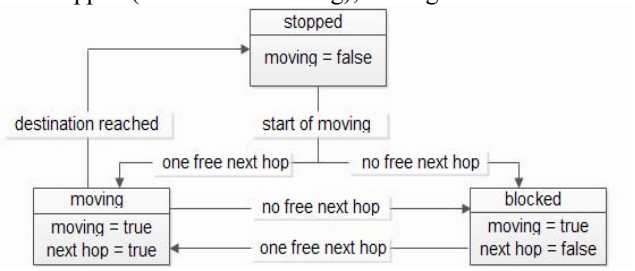


Figure 5. State machine for block motion.

When a block begins to move, it leaves the stopped state and enters in the moving state if at least one outgoing position from its current position is free, otherwise the block enters in the blocked state and waits until a valid hop becomes possible; in this last case, it enters in the moving state. A block finally returns to the stopped state when it reaches the destination position $P = D$ (see Fig. 5).

VI. SMART BLOCKS SIMULATOR

We present in this section, SBS, a multithreaded Java code Smart Blocks Simulator developed at LAAS-CNRS. The simulator SBS has permitted us to validate the proposed distributed iterative algorithm, to solve discrete trajectory optimization problems and to study in details communication schemes between blocks including control mechanisms to avoid message loops and network flooding. SBS permits one to simulate a modular surface with any size and different initial physical dispositions of blocks on the surface, to randomly set the computational input parameters, to launch the computation and graphically display its step by step course. Each block is managed by eight threads: a first group of four threads called Receive Threads (RT) and another group of four threads called Sensor Threads (ST). A RT continuously pools a Receive Buffer (RB) of the block and starts a new thread for processing each message retrieved from this buffer, implementing in this manner concurrent processing of different messages, e.g., the top Receive Buffer ($\mathrm{RB_{Up}}$) is always pooled by the Receive Thread called $\mathrm{RT_{Up}}$. A ST thread tracks one of the four neighboring positions in order to detect the presence/departure of a neighboring block to/from this position and to update the Neighbor Table. The blocks on the surface communicate with their neighbors via the Receive Buffers. The RB address of a block is known by the block and by the corresponding neighboring block, e.g., the $\mathrm{RB_{Up}}$ memory space of a block is shared with the neighboring block located on top of this block. The thread $\mathrm{RTh_{Up}}$ retrieves, treats and deletes from this space the messages written by a neighbor. Note that this memory organization does not require synchronization between two adjacent RT since one RT always reads and retrieves a message from an address of the receive buffer memory space while the other RT thread writes a message at a different address. Synchronization between the four receive threads of

a block is required only for processing the same type of message and for updating some local state variables like flags.

Memory organization is displayed on Fig. 6. For a typical block with four neighbors, data sent by neighbors according to the proposed communication scheme 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. 7). Note that the above memory organization follows the same design as the one of the multithreaded Smart Surface Simulator SSS developed to evaluate and validate distributed algorithms for differentiation of parts on a smart surface [12].

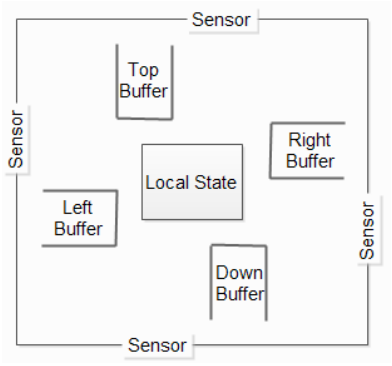


Figure 6. Basic block architecture with SBS.

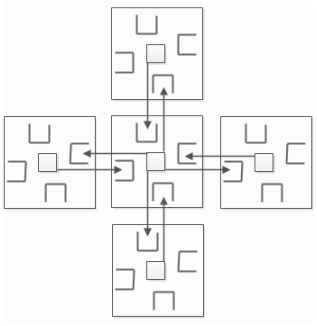


Figure 7. Block communication scheme with SBS.

VII. SIMULATION RESULTS

In order to validate the proposed distributed iterative algorithm, we have carried out some simulations with SBS. We present and analyze here some results. For facility of presentation, we concentrate here on a simple case of discrete trajectory optimization problem. The modular surface is composed of 10 blocks and the optimal trajectory necessitates 8 blocks. This small problem is solved in 8 iterations. Fig. 8a displays the initial state of the modular surface via SBS. Blocks are represented by blue squares; the green square corresponds to a block randomly chosen in order to start computation; the bottom and top gray squares represent the input position $(11,\ 3)$ of the parts to convey and their output position $(9,\ 8)$, respectively. Fig. 8b, 8c and 8d display, the disposition of blocks at the first, intermediate and last iterations, respectively. Fig. 9 shows the complete running trace of the distributed algorithm and displays the state of the modular surface. In Fig. 9, a small gray or dark square represents a block not yet elected or already elected and moved, respectively. A small red square is used to represent an outgoing position from a given place. An orange square represents a block elected at the current iteration which receives a CMR message in order to raise the next iteration from a block elected at the previous iteration (which is ready to move). A small blue square represents a block elected which is closest to a given outgoing position. Note that from IT^1 to IT^4, we are in the case where a block is close to the two outgoing positions; which leads this block to perform a random choice among these two outgoing positions. There are also some iterations at which only one outgoing position is valid since the other one is not in the rectangle bounded by I and O, e.g., IT^5 and IT^6. In this case, the block which is looking for an optimal position raises only one distributed election in order to know the identity of the block to which it will send a CMR message before the motion. Note also that when the block elected at iteration IT^6 receives the CMR message from the block elected at IT^5, it does not raises a new iteration since the only valid outgoing position from the position in CMR is the position of the output O. The last elected block sends a CCP message to its neighbors to order all nonelected blocks to move around the optimal path and completes the construction of the optimal path by moving directly to the output O.

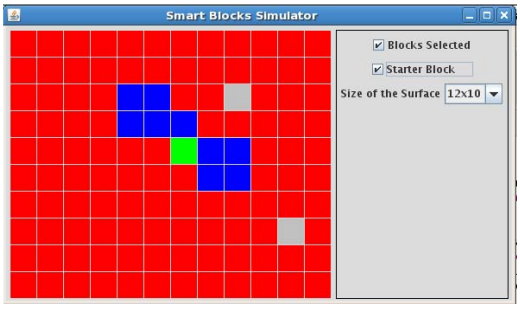
VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a distributed asynchronous iterative algorithm that solves a discrete trajectory optimization problem which occurs on a MEMS-based reconfigurable modular surface. The 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 entering point of parts and their exit point on the surface. Distributed election is implemented in order to obtain the block that can reach a given position on the surface with a minimum hop count; this block raises the next iteration before moving to its final position. The distributed elections are based on messages broadcasting with some control mechanisms to avoid message loops and network flooding. The proposed distributed approach presents the advantage to be scalable.

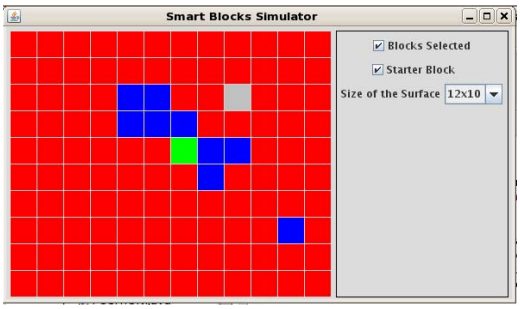
We have presented SBS, a multithreaded Java Smart Blocks Simulator that we have developed in order to validate the distributed algorithm. Finally, we have displayed and analyzed computational results obtained for the solution of a simple instance of the trajectory optimization problem.

The approach proposed in this paper is particularly useful to areas like semiconductors manufacturing, micro-mechanics and pharmaceutical industry since it is characterized by flexibility, scalability and optimality that are key issues in the development of future production lines.

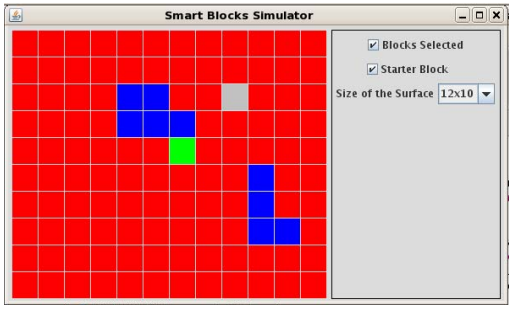
a. Initial state of the modular surface.



b. Surface state at IT^0



c. Surface state at IT^3



d. Surface state at IT^7

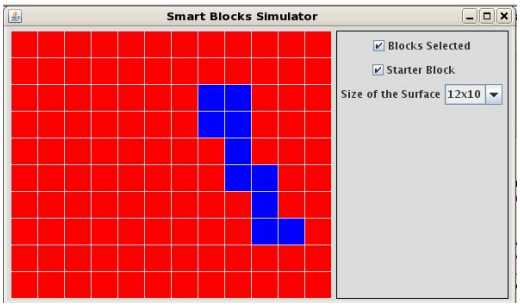


Figure 8. SBS modular surface windows

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

ACKNOWLEDGMENT

Part of this study has been made possible with the support of ANR-2011-BS03-005 grant.

REFERENCES

[1] "The Rules governing medicinal products in the European Union," chapter Good manufacturing practice guidelines, Eudralex, 2010.

[2] W.-M. Shen B. Salemi, M. Moll, "Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system," Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, 2006.

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

[4] Desnoyer M. Lipson H. Zykov V., Mytilinaios S, "Evolved and designed self-reproducing modular robotics," IEEE Transactions on Robotics, Vol. 23(2), 2007, pp. 308–319.

[5] D. Biegelsen et al., "Airjet paper mover," Proc. SPIE International Symposium on Micromachining and Micro fabrication, 4176-11, 2000.

[6] Y. Fukuta, Y. Chapuis, Y. Mita, 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 (4), 2006, pp. 912-926.

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

[8] J. Bourgeois et al., "Using a distributed intelligent MEMS modular and self-reconfigurable surface for fast conveying of fragile objects and medicinal products," Femto-st Technical report, submitted for publication, 2012.

[9] S. Konishi, H. Fujita, "A Conveyance System using air flow based on the concept of distributed micro motion systems," Journal of MicroelectroMechanical Syst., Vol. 3, 1994, pp. 54-58.

[10] A. Berlin and K. Gabriel, "Distributed MEMS: New challenges for computation," IEEE Computational Science and Engineering Journal, Vol. 4(1), 1997, pp. 12–16.

[11] K. Boutoustous et al., "Distributed control architecture for smart surface," in: Luo RC, Asaman H, editors, Proc. IROS, 23-rd IEEE/RSJ international conference on intelligent robots and systems. Tapei: IEEE Compute Society Press, 2010, pp. 2018-2024.

[12] D. El Baz et al., "Distributed part differentiation in a smart surface," Mechatronics, Vol. 22, 2012, pp. 522-530.

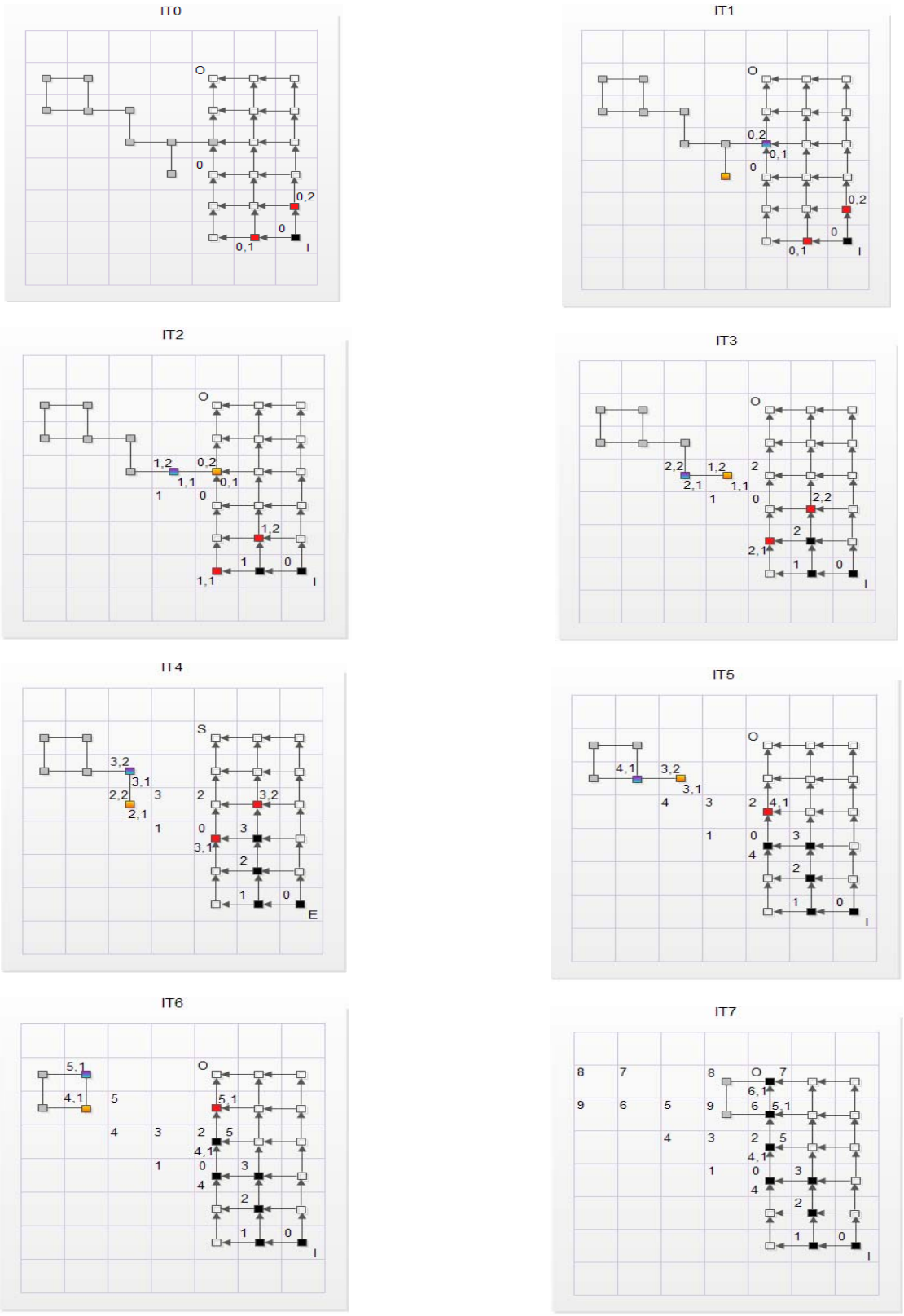


Figure 9. Running trace for an instance of discrete trajectory optimization problem.