Distributed part differentiation in a smart surface

Didier El Baz, Vincent Boyer, Julien Bourgeois, Eugen Dedu, Kahina Boutoustous

Distributed Part Differentiation in a Smart Surface#

Didier El Baz*, Vincent Boyer
CNRS ; LAAS ; 7 avenue du Colonel Roche, F-31077 Toulouse, France.
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
{elbaz, vboyer}@laas.fr

Julien Bourgeois, Eugen Dedu, Kahina Boutoustous
Laboratoire d'Informatique de l'Université de Franche-Comté 1 cours Leprince-Ringuet, 25200 Montbéliard France
{julien.bourgeois, eugen.dedu, kahina.boutoustous}@univ-fcomte.fr

***Abstract*—Distributed parts differentiation in a smart surface is considered. Synchronous and asynchronous distributed discrete state acquisition algorithms are proposed; their convergence is studied and implementation models are given. A distributed part differentiation method is proposed. A multithreaded Java Smart Surface Simulator (SSS) which runs on multi-core machines is presented. A series of computational results obtained with SSS is given and analyzed.**

Keywords—MEMS; smart surface; part differentiation, distributed algorithms; asynchronous algorithms; multi-core

1. INTRODUCTION

Micro-Electro Mechanical Systems (MEMS) actuator arrays with embedded intelligence, also referred to as smart surfaces, seem to have great potential impact for manipulating micro parts in many industrial areas like semiconductor industry and micromechanics (see [1, 2]).

The Smart Surface project aims at designing a micro robotics system (an array of fully integrated micro modules that are also referred to as cells) for conveying, sorting or positioning micro parts (see [3-5]). Each cell will contain a sensor, processing unit and actuators (see Fig. 1).

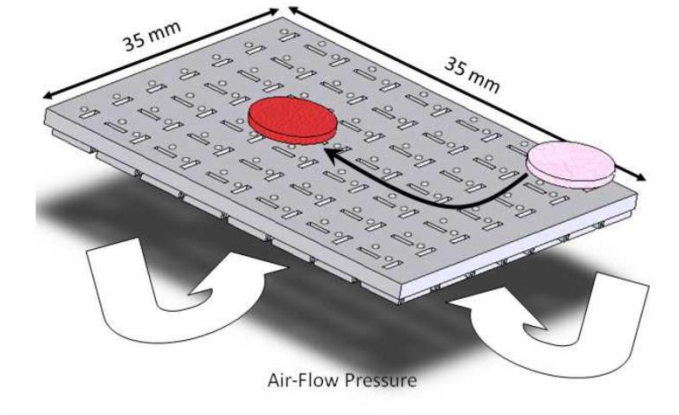

Fig. 1. General overview of the Smart Surface

Part of this study has been made possible by ANR grant ANR-06-ROBO-0009-03
* Corresponding author. Tel.: +33 561 336 303; fax: +33 561 336 411. E-mail address: elbaz@laas.fr

Distributed computing on dedicated architectures like smart surfaces is a source of a rich problematic mainly due to the scarcity of resources, e.g.: number of sensors (each part will recover a small number of sensors), memory size and computing power or the presence of faults. To the best of our knowledge, the literature on sorting and positioning micro parts in a low resolution context is almost nonexistent. For different approaches related to other applications in the low resolution context, we make reference to Ishida [6], for low resolution character recognition and Tabbone [7], for a novel approach based on the Radon transform for complex shapes identification (see also [8]).

In this paper, we propose an original approach for distributed part differentiation. Our method is decomposed into two phases. First, a distributed algorithm is used to obtain the discrete representation of parts on the smart surface as well as their position. Then, a distributed algorithm is used to differentiate the parts. The former phase is the so-called discrete state acquisition phase. The latter phase corresponds to the so-called differentiation phase that ends when each cell that is covered by a part has identified the type of the part.

We consider the case where many parts can be on the smart surface and extend the results in [9] (where we have made the assumption that there is at most one part at the same time on the smart surface). We give a mathematical model of discrete state acquisition and propose several distributed state acquisition algorithms. We consider synchronous and asynchronous iterative algorithms. We propose also simple initial points and give convergence results for the studied distributed algorithms. We propose stopping criteria in the synchronous case and in the asynchronous case.

We take opportunity of the high level of parallelism available on the array of micro modules to derive an original distributed algorithm that finally performs concurrent part differentiation. The techniques developed in this paper are particularly interesting when cells present faults or when parts are initially positioned any manner on the smart surface.

Finally, we present SSS, a multi threaded Java Smart Surface Simulator that has permitted us to evaluate and validate experimentally our distributed algorithms on multi-core machines.

Section 2 presents the smart surface. Section 3 deals with distributed discrete state acquisition. A first approach for part differentiation is presented in Section 4. Distributed part differentiation with gaps is proposed in Section 5. In Section 6, we present SSS, a multithreaded Java Smart Surface Simulator. Experimental results obtained with SSS are given and analyzed in Section 7. Conclusions are drawn in Section 8, where future work is also discussed.

2. THE SMART SURFACE

Assembly line workstations need to be fed with well-positioned and well-oriented parts. These parts are often jumbled and they need to be sorted and conveyed to the right workstation. To do so, the following operations must be performed on parts: identification, sorting, orienting, positioning and feeding. Among the most promising solutions to perform these tasks, is the combination of MEMS in order to form a micro robotics array.

There have been numerous projects on MEMS actuator arrays in the past and more particularly in the 1990's. These pioneering research works have developed different types of MEMS arrays that are based on actuators which are either pneumatic (see [10, 11]), magnetic or thermo bimorph, electrostatic or servo roller wheels (see [12]). Recent research works have been conducted in order to include sensors and to add intelligence to MEMS actuator arrays but these works either failed to propose solutions at a micro-scale or that are fully integrated (e.g. see [2]).

The objective of the Smart Surface project (see [13] and [14]) is to design a distributed and integrated micro manipulator based on an array of micro modules in order to realize an automated positioning and conveying surface. Each micro module will be composed of a micro actuator, a micro sensor and a processing unit. The cooperation of micro modules thanks to an integrated network will allow the system to recognize the parts and to control micro actuators in order to move and position accurately the parts on the smart surface. We consider small parts that cover a small number of micro modules and that are moved via air nozzles actuators (the rectangular holes on the front-side of Fig. 1). Air-flow comes through a micro valve in the back-side of the device and then passes through the nozzle (see Fig 2, for a front-side and back-side view of an actuator). The advantage of this solution is that the micro actuators, the most fragile part of the surface, are protected. Circle holes (see Fig. 1) are used by the micro sensors to detect the presence of the part on the surface.

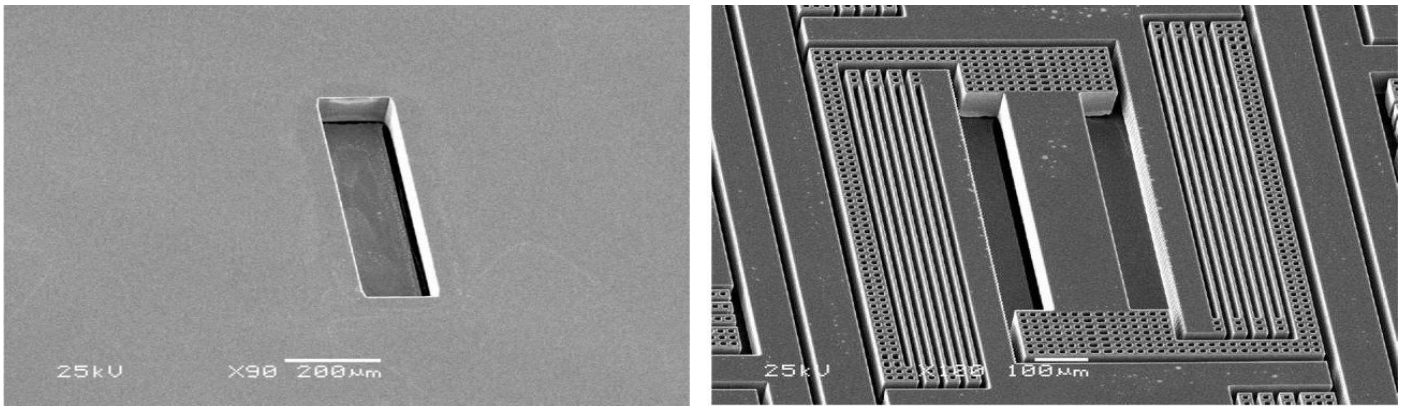

Fig. 2. Front-side and back-side of a micro-actuator

3. DISTRIBUTED DISCRETE STATE ACQUISITION

In this Section, we give a mathematical model for distributed discrete state acquisition and we derive several distributed algorithms. Cells are connected via a communication network. Each cell has at most 4 neighbors (see Fig. 2).

The problem is to obtain in a distributed way global knowledge of the discrete state of the smart surface, i.e. to obtain discrete representations of parts that lay on the smart surface and their exact position. In [9], we have made the assumption that there is only one part on the smart surface. In this paper, we extend this result to the case where several parts may lay on the Smart Surface.

3.1 Mathematical model of distributed state acquisition

Without loss of generality, we assume that there is only one sensor per cell. State acquisition can be modeled as the following fixed point problem: find $x^* \in E = \{0,1\}^{n^2}$ such that x^* is the smallest fixed point which satisfies:

$$x = F(x),$$

where n is the number of cells of the smart surface and F is a mapping from E into E. A vector $x \in E = \{0,1\}^{n^2}$ represents an augmented global state of the smart surface. This vector can be decomposed into sub vectors $x_i \in \{0,1\}^n, i \in \{1,\dots,n\}$, where x_i denotes the augmented local state of the ith cell. As a consequence, each cell needs only n bits in order to store its augmented local state (similarly, each cell needs at most $4n$ bits in order to store the augmented local state of its neighbors). The augmented local state of a given cell i corresponds to its current vision of the smart surface. The augmented local state can be decomposed into the actual local state of cell i, that is denoted by the scalar $x_{i,i}$ (if there is a part on cell i, then $x_{i,i} = 1$, otherwise we have $x_{i,i} = 0$) and the current knowledge cell i has of the smart surface, i.e. $x_{i,j},\ j \in \{1,\dots,n\},\ j \neq i$. The mapping F permits one to obtain a mathematical formulation of the state acquisition problem and to derive several useful distributed algorithms.

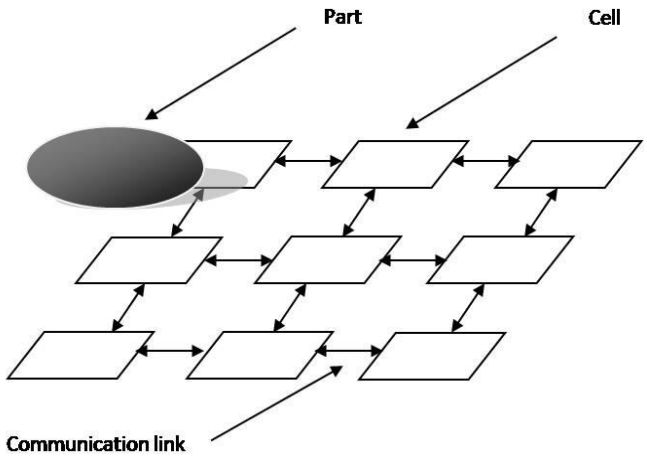

Fig. 3. Communication network of the smart surface

Cells make acquisition of the global state of the smart surface via data exchange, i.e. via messages they receive from their direct neighbors. Let $N(i)$ denote the set of all neighbors of cell i, the mapping F that can be decomposed the same way as vector x, is defined as follows:

$F_{i,i}(x) = x_{i,i}, i \in \{1, \dots, n\}$,

$F_{i,j}(x) = x_{i,j} + x_{j,j}, if\ j\ \in N(i), i \in \{1, \dots, n\}$,

$$F_{i,l}(x) = x_{i,l} + \sum_{j \in N(i)} x_{j,l},\ i, l \in \{1, \dots, n\}, l \notin N(i),$$

where the operations $+$ and $\sum_{j \in N(i)}$ are defined as follows: let $a, b \in \{0,1\}$, $a + b = 0, if\ a = 0\ and\ b = 0$, $a + b = 1$, otherwise; similarly, $\sum_{j \in N(i)} x_{j,l} = 1\ if\ at\ least\ one\ x_{j,l} = 1$ and $\sum_{j \in N(i)} x_{j,l} = 0$ otherwise.

Definition 1: Distributed synchronous state acquisition can be described via the following successive approximation method (this method is also referred to as a discrete iteration or an iteration on a product of finite sets):

$x^{k+1} = F(x^k), k = 0,1, \dots$

Where the initial approximation x^0, is chosen as follows:

$x_{i,j}^0 = 0, if\ j \neq i, i, j \in \{1, \dots, n\}$,

$x_{i,i}^0 = 1, if\ there\ is\ a\ part\ on\ cell\ i, i \in \{1, \dots, n\}$,

$x_{i,i}^0 = 0, if\ there\ is\ no\ part\ on\ cell\ i, i \in \{1, \dots, n\}$.

Remark 1: Clearly, x^* is not the only fixed point for F. It is possible that the sequence $\{x^k\}$ generated by a discrete iteration starting from $x \in E, x \neq x^0$ remains stationary at a given vector $x', x' \geq x^*$, where the order relation is component wise, i.e. $x'_{i,j} \geq x^*_{i,j}, \forall i, j\ \in \{1, \dots, n\}$. A simple illustration can be obtained by choosing $x \in E$such that $x_{i,j} = 1, \forall i, j \in \{1, \dots, n\}$. In this case, we clearly have: $x = F(x)$. However, the values of the component $x_{i,i} = 1, \forall i \in \{1, \dots, n\}$ derived from x do not correspond to the actual local state of the cell of the smart surface, unless the part covers completely the surface. This shows the importance of choosing a good initial approximation in order to converge to a fixed point that makes sense in term of state acquisition. We shall see in the sequel that the above defined initial vector x^0, which is in fact a sub solution, i.e. which satisfies $x^0 \leq x^*$(the inequality being considered component wise), is a good starting point in order to converge to the solution x^* of the fixed point problem that makes sense in term of state acquisition.

The reader is referred to Robert [15] for a study on mathematical and algorithmic aspects of discrete iterations.

Theorem 1: The mapping F is monotone.

Proof: from the definition of the mapping F, we clearly have $F(x) \leq F(x'), \forall x, x' \in E\ such\ that\ x \leq x'$; Q.E.D.

Theorem 2: The distributed discrete iteration $\{x^k\}$ converges to x^*.

Proof: from the definition of mapping F, we have

$x^0 \leq x^1 = F(x^0)$.

Moreover, we have

$x^k \leq x^{k+1} = F(x^k), \forall k = 1,2, \ldots$

$x^k \leq x^*, \forall k = 1, 2, \ldots$

Since the mapping F is monotone, $x^0 \leq x^*$ and $x^* = F(x^*)$. We denote by d a discrete distance on $\{0, 1\}$. We have:

$d(x_{i,j}, x'_{i,j}) = |x_{i,j} - x'_{i,j}|, \forall x, x' \in E, \forall i, j \in \{1, \ldots, n\}$.

Let d^m be the Manhattan distance on E, we have:

$$d^m(x, x') = \sum_{i=1}^{n} \sum_{j=1}^{n} |x_{i,j} - x'_{i,j}|, \forall x, x' \in E.$$

We note that $d^m(x, x')$ is finite for any $x, x' \in E$. As a consequence, the distributed discrete algorithm $\{x^k\}$ converges monotonically to x^* in finite number of iterations; Q.E.D.

Let us denote now by d' the largest Manhattan distance of the smart surface.

Theorem 3: The number of iterations of the discrete algorithm is bounded by $d' + 1$.

Proof: The distributed discrete iteration starting from $x^0 \in E$, generates a monotone sequence $\{x^k\}$ of vectors of E. Time after time, each cell makes acquisition of the augmented local state of its neighbors via messages it receives from them and combines these augmented states with its own augmented state in order to produce an updated augmented state, i.e. a more accurate vision of the state of the mart surface. At each new iteration, cells gain a more accurate vision of the actual discrete state of the smart surface by a unit of distance. This corresponds to a link between two cells along each direction. Finally, the sequence converges to the fixed point $x^* \in E$ which is such that $x^*_{i,j} = x^*_{j,j}, \forall i, j \in E$; this shows that at convergence, all augmented local states are similar and that all cells have the same vision of the global state of the smart surface). Q.E.D.

For a rectangular smart surface with size $a \times b$, at most $(a - 1) \times (b - 1)$ sequences of communications are necessary to make acquisition of local states and at most $(a - 1) \times (b - 1) + 1$ iterations are necessary to obtain the solution.

In [9], we have proposed a local stopping test that permits one to reduce the number of iterations in situations where we assume that there is at most one part on the smart surface.

3.2 Implementation of the distributed algorithm

For simplicity of notation, we denote in the sequel by N_i the set of neighbors of node i. The jth neighbor of node i is denoted by $n_i(j)$. The behavior of the distributed synchronous discrete algorithm can be represented as follows.

For *i* from 1 to *n* do

  For *k* from 1 to $d' + 1$ do

    $x_i^k := F_i(x^{k-1})$

    For *j* from 1 to $card(N_i)$ do

      send x_i^k to $n_i(j)$

    End do

    For *j* from 1 to $card(N_i)$ do

      receive x_j^k from $n_i(j)$

    End do

  End do

End do

Distributed synchronous discrete algorithm

3.3 Distributed asynchronous algorithms

In the above subsection we have presented a first model of distributed state acquisition in the synchronous case. We present now a mathematical model in the more general asynchronous context where each cell can perform updating phases at its own pace,

i.e. computation can be done without order nor synchronization. We derive convergence results by using the general convergence theorem of Bertsekas (see [16]). We propose also a stopping method.

We assume that there is a set of times $T = \{0,1,2,...\}$ at which one or more sub vectors $x_i, i \in \{1,...,n\}$, of vector x are updated by some cells. We denote by $T(i)$ the subset of times at which the sub vector x_i is updated. Let $L_i = \{s_{1,i}(k),...,s_{n,i}(k)\}$ be the subset of labels used during the updating phases of cell i with:

$0 \leq s_{j,i}(k) \leq k, \forall\, j, i \in \{1,...,\mathrm{n}\}, \forall k \in T(i)$.

We assume that $\lim_{k\to\infty} s_{j,i}(k) = +\infty, \forall j, i \in \{1,...,n\}$, this assumption guarantees that new values of the components of the sub vectors are used as computations go on. We also assume that the sets $T(i)$, $i = \{1,...,n\}$, are infinite; this assumption guarantees that no component of the iterate vector is abandoned forever. In particular, cells will not stop their computations before convergence.

We denote by *L* the set of labels used during the computations performed by the different cells.

Definition 3: Distributed asynchronous state acquisition can be described via the following successive approximation method denoted by (F, x^0, T, L), where x^0 is the initial approximation defined in subsection 2.1.

$x_i^{k+1} = F_i\left(x_1^{s_{1,i}(k)},...,x_n^{s_{n,i}(k)}\right), \forall k \in T(i),$

$x_i^{k+1} = x_i^k, \forall k \notin T(i),$

Theorem 4: The distributed asynchronous discrete iteration (F, x^0, T, L) converges to x^*.

Proof: In order to show convergence of the asynchronous algorithm (F, x^0, T, L) we build a sequence of level sets which satisfies the conditions of the general asynchronous convergence theorem of Bertsekas, see page 431 in [16].

Let $E^0 = \{x \in E \,/\, x^0 \leq x \leq x^*\}$, we define the sets

$E^k = \{x \in E \,/\, F^k(x^0) \leq x \leq x^*\}$.

The so-called synchronous convergence condition of Bertsekas

$F(x) \in E^{k+1}, \forall k, \forall x \in E^k$, and every limit point of $\{x^k\}$ is a fixed point of *F* if $x^k \in E^k, \forall k$, is satisfied. This result follows from Theorem 2, the monotone property of mapping *F* and the fact that x^* is the smallest vector such that $x = F(x)$.

The so-called box condition of Bertsekas, i.e.:

$E^k = E_1^k \times E_2^k \times ... \times E_n^k, \qquad \forall k = 0,1,2,...$

is also satisfied since the level sets E^k are Cartesian products of subsets $\{0,1\}$. As a consequence, the general asynchronous convergence theorem of Bertsekas applies. Q.E.D.

We note that the sequence of nonempty subsets E^k satisfies:

$E^\infty \subset \cdots \subset E^{k+1} \subset E^k \subset \cdots \subset E^0$,

and

$E^\infty = \{x^*\}$.

The reader is also referred to Radid [17] for various results related to asynchronous discrete iterations.

Among the many interests of distributed asynchronous iterations, one can quote the better efficiency of the algorithms since each cell goes at its own pace and there is no waiting time for synchronization. This is particularly true in the case of monotone convergence, where the use of last updates permits always one to improve the iterate vector. One can quote also fault tolerance since distributed asynchronous iterations tolerate some messages losses (see [16]).

In the case of a cell fault, a distributed asynchronous algorithm may end with an approximation of the solution that is slightly different from x^*, however, we shall see in the next Section that the gap-based techniques developed in this study permit one to overcome this difficulty . Moreover, there will be no deadlock with such an asynchronous algorithm.

3.4 Implementation of distributed asynchronous algorithms

In this subsection, we show how asynchronous algorithms have been implemented. We consider also convergence detection of asynchronous algorithms. Several procedures can be used in order to detect convergence of distributed asynchronous discrete iterations. One can use for example the Dijkstra and Scholten procedure [18] (see also [19, 20]). The reader is also referred to El Baz [21] for a method based on level sets. The procedure in [18] relies on generation of activity graph and acknowledgement of messages. Initially, only one cell is active, i.e. the so-called root that is denoted by *R*. The cell *R* starts computation and sends

messages to its neighbors; these messages activate the neighbors that become the so-called sons of R and so on. All cells become eventually active. All messages are acknowledged at once but activation messages of father that are acknowledged only when a son becomes inactive. The activity graph moves on according to the messages received and satisfaction of the conditions: $x_i^{k+1} = x_i^k$. A cell sends messages to its neighbors if and only if it is active and the above condition is not satisfied. Finally, the algorithm stops when the cell R stops; i.e. all local stopping criteria are satisfied and there is no message in transit in the system. This type of convergence detection method is quite natural in the context of discrete iterations since it is not necessary to modify the distributed asynchronous iterative algorithm so that it converges in finite time.

Let us denote by Active(i) the logical variable that stores the behavior of the ith cell: if Active(i) is True, then the ith cell performs computation. If Active(i) is False, then the ith cell does nothing. Initially, all cells are inactive, but R. A cell becomes active when receiving a message. A cell i becomes inactive when the following extended local asynchronous stopping criterion is satisfied.

Definition 4: The extended asynchronous local stopping test is given by: $x_i^k = x_i^{k-1}$ and all cells activated by cell i are inactive.

All cells can be activated many times but the root, R, which is active only once. Finally, the algorithm stops when R becomes inactive. In the sequel, we shall denote by isend and ireceive, respectively, non blocking send and receive, respectively. These communication primitives permit one to implement asynchronous communication.

For i from 1 to n do

While Active(R) = True

If Active(i) = True then

$k := k + 1$

$x_i^k := F_i\left(x_1^{s_{1,i}(k)}, \dots, x_n^{s_{n,i}(k)}\right)$

If $x_i^k \neq x_i^{k-1}$, then

For j from 1 to $card(N_i)$ do

isend x_i^k to $n_i(j)$

End do

End if

For j from 1 to $card(N_i)$ do

ireceive x_i^k to $n_i(j)$

End do

End While

End do

Distributed asynchronous discrete algorithm

Formal proofs of validity for this type of algorithm including convergence detection method have been established in [19, 20].

4. A FIRST APPROACH FOR DISTRIBUTED PART DIFFERENTIATION

We assume that there are a limited number of types of parts. We shall consider in this paper that we have three types of parts. Thus, any part on the smart surface belongs to one of these three classes. The algorithms we propose, nevertheless, work well with more classes.

This Section presents the first approach we have proposed for part differentiation in a low resolution context and when there is only one part on the smart surface (see [3]). This approach is based on the computation of criteria. These criteria are basically contour-based differentiation criteria like number of components of vector x_i with value 1 such that there exists $x_j = 0, j \in N(i)$ or region-based criteria, like number of components of vector x_i with value 1, i.e. surface like criteria, or maximum length between 1 of the part. The different criteria used in this paper are detailed in [3] (see also [22]).

The approach presented here is particularly interesting when one aims at determining if part differentiation is possible or not. This is why we also refer to this approach as total differentiation. This approach consists of two stages that are detailed below.

- offline stage: during this stage a database that contains the values of criteria used to differentiate parts is produced. Only a limited number of parts that will be the so-called reference parts is considered. The values of the criteria are stored at each cell.
- online stage: during this stage cells try to differentiate the part on the Smart Surface by comparing the criteria values of the current part on the smart surface with the database.

4.1 Offline stage

This stage permits one to associate to each reference part a set of criteria values. For each reference part, the following phases are executed in sequence:

- the reference part is rotated one degree with respect to its centre and translated with s/10 cells, where s denotes the width of the rotated part;
- a matrix associated with sensor values is generated, the matrix fits the Smart Surface, i.e. there is one entry per sensor, an entry of the matrix is equal to 1 if its associated sensor is covered by a part, it is equal to 0 otherwise;
- sub-matrices of the initial matrix without rows and columns that contain only zeros are generated, these sub-matrices are the so-called masks, multiple copies of the same mask are discarded;
- values of the several criteria are calculated for all masks of each reference part.

These criteria values form the database to be used as input of the online stage.

4.2 Online stage

This stage takes place after the discrete state acquisition phase. The aim of this stage is to differentiate, in real-time, a part on the Smart Surface.

Once each cell of the smart surface has made acquisition of the binary representation of the part on the smart surface, it computes the criteria values. These values are compared to the criteria values computed at the offline stage in the database. If there is a criterion value or a combination of criteria values that matches exactly the one in the database, then the part is considered to be differentiated. If no such correspondence is found, then no decision is taken.

5. DISTRIBUTED PART DIFFERENTIATION WITH GAPS

In this Section, we propose an original distributed part differentiation method based on gaps. As in the previous Section, we assume that there are a limited number of types of parts (we recall that only three types of parts are considered in this paper). Our algorithm, nevertheless, works well with more classes.

We take benefit of the high level of parallelism available on the array of micro modules to derive a distributed algorithm. Modules will compute concurrently several contour-based or region-based criteria related to the part that covers them. Decisions that are based on the value of the criteria are taken concurrently. The differentiation phase ends when each cell that is covered by a part has identified the type of the part.

We detail now the decision process. We note that the value of the different criteria can vary according to the orientation and position of the part on the smart surface. In the case of part rotation, for example, we have observed that the surface of 3x3 square, where the unit of distance is the length of a cell, can vary from 9 to 13, according to the orientation of the part on the smart surface, see Fig. 3 obtained with SSS, a multithreaded smart surface simulator that will be detailed in the next Section. Fig. 4 shows a view of an actual part on the smart surface and its discrete representation.

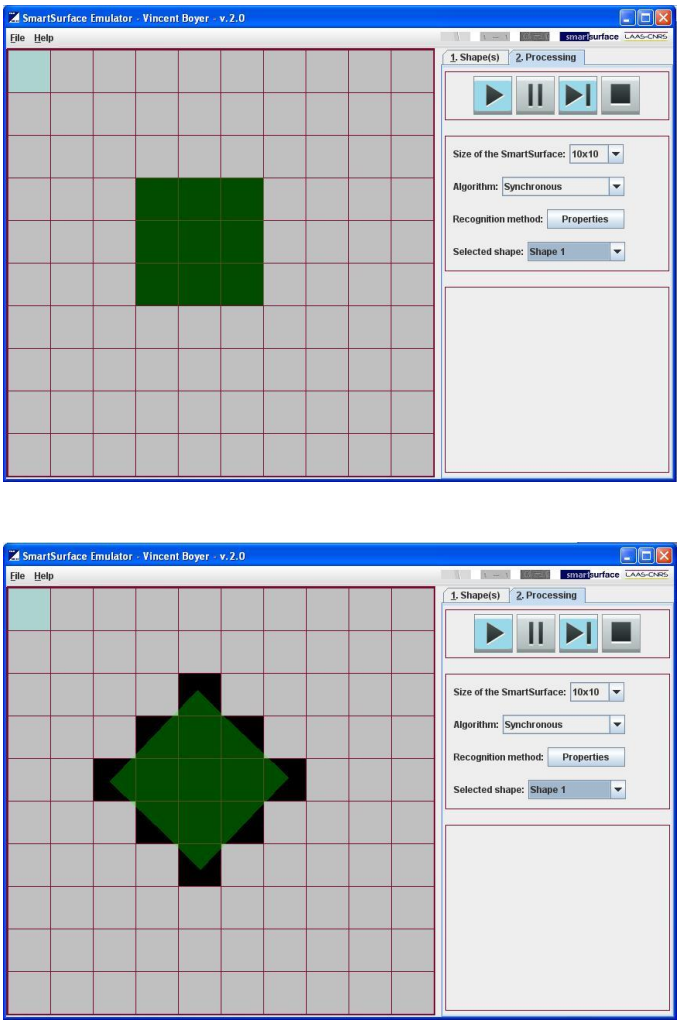

Fig 4. Examples of different surface values for the same square with different orientations (SSS screenshots)

In the sequel, we present two new approaches for part differentiation.

The first approach relies on the use of a single reference position for each part. Each cell compares computed values of the criteria for the current position of the part on the smart surface with values of criteria of each part obtained for a single reference position (those values are stored in a database of registered parts). We propose to compute gaps between the measured criteria and the criteria value of registered parts for differentiation purpose. This method is particularly interesting when some cells present faults.

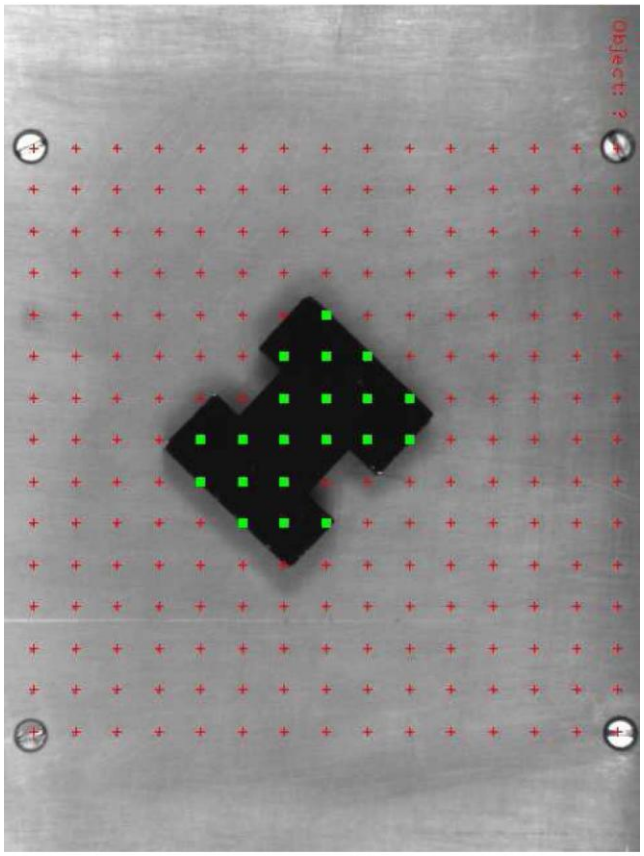

Fig. 5. View of an actual H shape part on the Smart Surface and its discrete representation (courtesy of Femto-ST)

Another particularity of the first approach is to consider only a subset of well known criteria like surface or perimeter of the part. The criteria that amplify tiny differences between parts, like product of the differences between consecutive columns and consecutive line are discarded. In the sequel, the number of criteria considered will be denoted by q. Let m denote the number of different parts. Let $r_i(j)$ denote the reference value of ith criterion of the jth part (those values are computed offline). Let c_i denote the value of the ith criterion of the part on the smart surface. Each cell computes the following gaps:

$$g(j) = \frac{1}{q}\sum_{i=1}^{q}\left|\frac{r_i(j)}{c_i} - 1\right|, j \in \{1, \dots, m\}.$$

The second approach relies on the use of a set of reference positions for the different parts on the smart surface. We take into account rotations of parts with one degree increments. Without loss of generality and for symmetry reason, only rotations from 1 up to 45 degrees can be considered. Let $D = \{1, \dots, 45\}$. We denote by $r_i^d(j)$ the reference value of ith criterion of the jth part rotated by d degrees; those values are also computed offline. We denote by $C(j)$ the set of all reference values for part j, $C(j) = \left\{\left(r_1^d(j), \dots, r_q^d(j)\right), d \in D\right\}$. Each cell computes the following gaps.

$$g'(j) = min_{d\in D}\left\{\frac{1}{q}\sum_{i=1}^{q}\left|\frac{r_i^d(j)}{c_i} - 1\right|\right\}, j \in \{1, \dots, m\}.$$

We note that the former gap, g, presents the advantage to require a limited amount of memory and a small computing time, while the latter gap, g', permits one to expect better differentiation of parts, particularly in the case where parts can have any orientation on the smart surface.

Decision making concerning pattern recognition at each cell relies on the respective values of the gaps. The pattern j that is chosen corresponds to the gap $g(j)$ or $g'(j)$ that is the closest to zero. We note that all cells make the same computation concurrently and thus take the same decision.

Concurrent decision making based on gaps is particularly interesting with respect to fault that may occur on the smart surface, i.e. sensor failures or parts positioned any manner on the smart surface. The use of gaps is also interesting with respect to recognition of pattern slightly modified, i.e. parts that present tiny faults.

6. SSS, A MULTITHREADED SMART SURFACE SIMULATOR

We present now SSS, a smart surface simulator developed at LAAS-CNRS. The simulator SSS has permitted us to evaluate distributed synchronous and asynchronous state acquisition algorithms and concurrent pattern recognition methods. SSS is a multithreaded Java code that runs on multi-core machines.

SSS has permitted us to validate experimentally the distributed algorithms and to study in detail communications between cells, stopping criteria and the efficiency of the proposed methods. The reader is referred to the site [5] for some demos with SSS.

SSS permits one to build a smart surface that has any size and different basic patterns like squares, rectangles, L shapes, I shapes and so on, that will become references or that will correspond to a given part. SSS permits one also to place the generated patterns everywhere on the smart surface. It is possible to rotate shapes on the smart surface and to introduce sensors faults (see Fig. 3, for a square shape). SSS allows one to choose a synchronous or asynchronous distributed state acquisition algorithm, to carry it out and to display dynamically the augmented local state of any cell (see right window of Fig. 5 that corresponds to iteration 2). One can have a dynamic view on the activity graph of the smart surface, i.e. one can see the cells that are active (those cells who are updating their augmented local state). One can also choose particular criteria and a differentiation method, e.g. gaps based methods or differentiation methods studied in [22]. Finally, one can display the results of the pattern recognition phase for the different criteria selected. SSS permits one also to display some statistics.

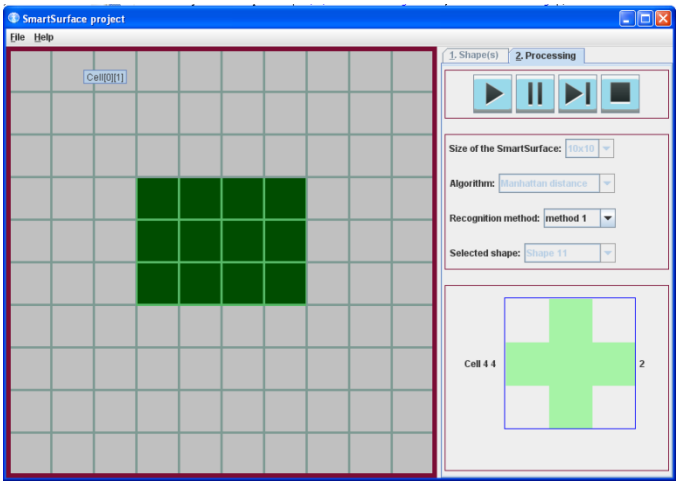

Fig. 6. SSS smart surface window (left) and "extended" local state window of a given cell at iteration 2 (right)

In SSS, each cell is managed by a thread. Cells communicate with their neighbors via buffers (see Fig. 6 and 7). Buffers are controlled by flags which enable or disable memory access, e.g. writing. This permits one to handle memory conflicts and to implement synchronization between threads.

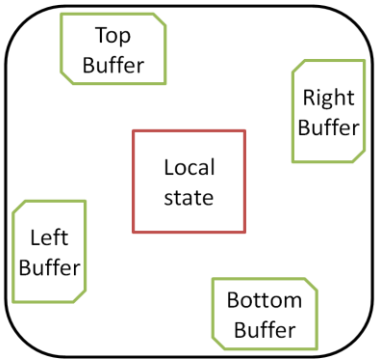

Fig. 7. Basic cell architecture with SSS

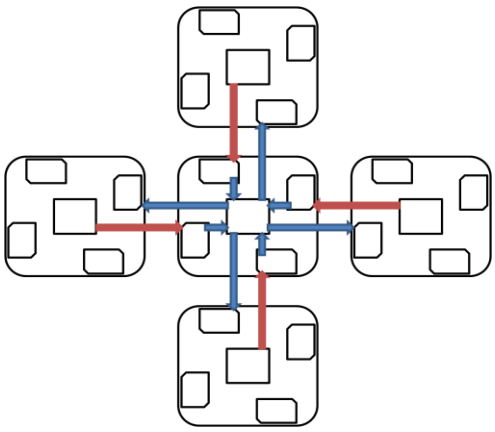

Fig. 8. Cell communication scheme with SSS

Memory organization, displayed in Fig. 6, is given as follows. For a typical cell with four neighbors, data sent by neighbors according to communications scheme presented in Fig. 7 are stored in a dedicated buffer, e.g. top buffer, for neighbor that is above the considered cell, right buffer for neighbor that is situated on the right of the cell … Finally, local state is used to store the current value of the augmented local state. This organization presents great flexibility in order to study different algorithms for state acquisition. In particular, it permits one to implement easily asynchronous schemes of computation since in this case neighbors can freely exchange data with a cell without synchronization. It permits nevertheless one to implement synchronous schemes of computation whereby neighbors must be granted permission to access buffers.

7. TESTS

The multithreaded Java smart surface simulator SSS has been carried out in parallel on a multi-core machine with 3.0GHz Quad-Core Intel Xeon processor.

In this Section, we compare a first series of results obtained with SSS for the gap methods proposed in Section 5 and the total differentiation method presented in Section 4. We have considered three parts: a square, the so-called Sq part, an L shaped part and a I shaped part. All parts have been placed randomly 200 times, leading to 200 draws with SSS. We have classified the criteria according to their performance. For each draw, we have computed the values of 17 criteria and we have applied the total differentiation method of Section 4 which gives the differentiation rate. The criteria were afterwards classified according to the differentiation rate and only the criteria with the best differentiation rates were selected.

Table 1 displays results for the best two criteria: S and A, respectively, were S denotes the surface and A the sum of angles of type "V", respectively (see [22]). For 200 draws, we note that the criterion S has correctly differentiated the part Sq in 37% of cases.

Criteria	Sq	I	L	Average
S	37.00%	52.00%	57.30%	48.76%
A	33.50%	40.50%	48.50%	40.83%

TABLE 1
DIFFERENTIATION RATES FOR THE CRITERIA A AND S,TOTAL DIFFERENTIATION

In the second test, we are interested in part differentiation via a combination of the criteria S and A. Table 2 shows the differentiation rate obtained by using the criteria combination. We note that the part Sq was correctly differentiated in 59% of the cases; which is better than with S or A alone. The average differentiation rate was increased from 48.76% (for S alone) and 40.83% (for criterion A alone) up to 67.16% for the combination of A and S.

Criteria	Sq	I	L	Average
A and S	59,50%	74,00%	68,00%	67,16%

TABLE 2

Differentiation rates for criteria combination, total differentiation

We compare now the results of the above differentiation method with those obtained with the methods based on gaps introduced in Section 5 of this paper. Table 3 displays the results obtained with the gaps g.

Criteria	Sq	I	L	Average
S	100%	99.00%	98.00%	99.00%
A	100%	99.00%	78.00%	92.33%
A and S	100%	100%	96.50%	98.83%

TABLE 3
Differentiation rates with the first gap

Table 4 gives the results obtained with the gaps g'.

Criteria	Sq	I	L	Average
S	100%	99.00%	98.00%	99.00%
A	100%	99.00%	78.00%	92.33%
A and S	100%	99.50%	79.00%	92.83%

TABLE 4
Differentiation rates with the second gap

These results show that the methods based on gaps g and g' improve the differentiation rate in the following cases:
- with a single criterion, e.g. the criterion S, the differentiation rate is improved from 48.76% (see first row of Table 1) to 99% for g (see first row of Table 3) and 99% for g' (see first row of Table 4).
- with several criteria, the combination of criteria also improves the differentiation rate from 67.16% (see first row of Table 2), to 98.83% (see third row of Table 3) and 92.83% (see third row of Table 4), respectively, for g and g', respectively.

We conclude that the differentiation methods based on gaps give better results than the total differentiation method when parts can have any orientation on the smart surface. Additionally, one criterion alone may be sufficient to reach almost 100% differentiation rate.

8. CONCLUSIONS AND PERSPECTIVES

In this paper, we have considered a smart surface for conveying, sorting or positioning micro parts. We have given a mathematical model for discrete state acquisition. We have proposed several distributed synchronous and asynchronous algorithms, some stopping criteria and we have established convergence results for the studied methods. We have also proposed a distributed part differentiation method based on gaps. Finally, we have presented SSS, a multithreaded Java Smart Surface Simulator, that we have developed in order to evaluate and validate distributed algorithms and we have displayed and analyzed a first series of results for randomly generated instances with SSS.

Future directions of research concern fault detection and solutions that propose degraded but everlasting behavior particularly in the synchronous case.

We shall consider concurrent pattern recognition methods that exploit smart surface natural parallelism, e.g. each cell i can apply a mapping $T_i$to x_i and compute criteria. Typical mappings $T_i$can be some rotations. This kind of preconditioning can enrich pattern recognition.

We shall also study combined part differentiation and part motion; it may be efficient to recognize a part while moving it on the smart surface. The reader is referred to [23] for a first study.

It may also be interesting to derive pattern recognition techniques that are not criteria based and which exploit directly the part code.

Finally, implementation on the distributed smart surface must be made in order to complete the study.

9. ACKNOWLEDGEMENTS

The authors would like to thank Femto-ST for photos of the parts and information regarding the smart surface.

REFERENCES

[1] D. Biegelsen et al. Airjet paper mover, in SPIE International Symposium on Micromachining and Micro fabrication, 4176-11, Sept. 2000.

[2] 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, 15 (4) (2006) 912-926.

[3] K. Boutoustous, E. Dedu, J. Bourgeois, A framework to calibrate a MEMS sensor network, in Proc. of the International Conference on Ubiquitous Intelligence and Computing, Brisbane Australia, July 2009, Lecture Notes in Computer Science, Springer, Berlin, 2009, 136-149.

[4] http://www.smartsurface.cnrs.fr/ Smart Surface ANR project, 2010.

[5] http://spiderman-2.laas.fr/SMART-SURFACE/ Smart Surface project at LAAS-CNRS, Toulouse, 2010.

[6] H. Ishida et al. Recognition of low resolution characters by a generative learning method, in Proc. of the 1st International Workshop on Camera-Based Document Analysis and Recognition, pp. 45-51.

[7] S. Tabbone, L. Wendling, and J.-P. Salmon. A new shape descriptor defined on the Radon transform, Computer Vision and Image Understanding, 102 (1), (2006), 42-51.

[8] J. Kare, Backyard star wars, IEEE Spectrum, May 2010, 26-29.

[9] D. El Baz, V. Boyer, J. Bourgeois, E. Dedu, K. Boutoustous, Distributed discrete state acquisition and concurrent pattern recognition in a distributed MEMS-based Smart Surface, in Proceedings of the dMEMS Conference, IEEE, Besançon, France, 28-29 Février 2010.

[10] H. Fujita. Group work of microactuators, In International Advanced Robot Program Workshop on Micromachine Technologies and Systems, Tokyo, Japan, October 1993, pp. 24–31.

[11] K.S.J. Pister, R. Fearing, and R. T. Howe. A planar air levitated electrostatic actuator system, in IEEE Micro Electro Mechanical Systems. An Investigation of Micro Structures, Sensors, Actuators, Machines and Robots, (1990) pp. 61–71.

[12] K. F. Bohringer, V. Bhatt, B. R. Donald and K. Y. Goldberg. Algorithms for sensorless manipulation using a vibrating surface, Algorithmica, 26 (3-4) (2000) 389–429.

[13] N. Le Fort-Piat et al. Smart Surface based on autonomous distributed micro robotic systems for robust and adaptive micromanipulation, ANR Smart Surface Project Proposal, 2006.

[14] L. Matignon, G. Laurent, N. Le Fort-Piat, Design of semi-decentralized control laws for distributed-air-jet micromanipulators by reinforcement learning, IROS, 2009, 3277-3283.

[15] F. Robert, Discrete Iterations, a Metric Study, Springer Series in Computational Mathematics, Springer Verlag, Berlin, 1986.

[16] D.Bertsekas, J. Tsitsiklis, Parallel and distributed iterative algorithms: a selective survey, Automatica, 25 (1991) 3-21.

[17] A. Radid, Itérations Booléennes et sur des ensembles de cardinal fini, Ph. D. of the Franche-Comté university, 2000.

[18] E. W. Dijkstra and C. S. Scholten, Termination detection for diffusing computation, Information Processing Letters, 11 (1980) 1-4.

[19] D. Bertsekas, J. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Athena Scientific, 1997.

[20] D. El Baz, An efficient termination method for asynchronous iterative algorithms on message passing architecture, in Proc. of the International Conference on Parallel and Distributed Computing Systems, Dijon, Vol. 1, 1996, pp. 1-7.

[21] D. El Baz, A method of terminating asynchronous iterative algorithms on message passing systems, Parallel Algorithms and Application, 9 (1996) 153-158.

[22] K. Boutoustous, E. Dedu, J. Bourgeois, An exhaustive comparison framework for distributed shape differentiation in a MEMS sensor actuator array, in Proc. International Symposium on Parallel and Distributed Computing (ISPDC), IEEE Computer Society Press, Kraków, Poland, 2008, pp. 429-433.

[23] K. Boutoustous et al., Distributed control architecture for smart surfaces, LIFC/Femto-ST report.