Mechatronics xxx (2011) xxx–xxx

Contents lists available at ScienceDirect

Mechatronics

journal homepage: www.elsevier.com/locate/mechatronics

Distributed part differentiation in a smart surface ☆

Didier El Baz [a,b,*], Vincent Boyer [a,b], Julien Bourgeois [c], Eugen Dedu [c], Kahina Boutoustous [c]

[a] CNRS, LAAS, 7 Avenue du Colonel Roche, F-31077 Toulouse, France
[b] Université de Toulouse, UPS, INSA, INP, ISAE, LAAS, F-31077 Toulouse, France
[c] Laboratoire d'Informatique de l'Université de Franche-Comté 1, Cours Leprince-Ringuet, 25200 Montbéliard, France

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
MEMS
Smart surface
Part differentiation
State acquisition
Distributed algorithms

ABSTRACT

Distributed differentiation of parts 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. Distributed part differentiation methods are 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.

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 microparts in many industrial areas like semiconductor industry and micromechanics (see [1,2]).

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

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 microparts 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 et al. [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 in order 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 referred to as the discrete state acquisition phase. The latter phase corresponds to the differentiation phase that ends when all cells that are covered by a part have reached a decision about the type of that part.

We consider here the case where many parts can be on the smart surface and we extend the results in [9] (where the assumption was made that there was 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 micromodules in order to derive an original distributed algorithm that performs concurrent part differentiation. The techniques developed in this paper are particularly interesting when parts are positioned any manner on the smart surface. As we shall see in the sequel, the proposed techniques are also attractive when some sensors have a fault, e.g. the sensors do not detect a part.

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

☆ Part of this study has been made possible by ANR Grant ANR-06-ROBO-0009.
* Corresponding author at: Université de Toulouse, UPS, INSA, INP, ISAE, LAAS, F-31077 Toulouse, France. Tel.: +33 561 336 303; fax: +33 561 336 411.
E-mail addresses: elbaz@laas.fr (D. El Baz), vboyer@laas.fr (V. Boyer), julien.bourgeois@univ-fcomte.fr (J. Bourgeois), eugen.dedu@univ-fcomte.fr (E. Dedu), kahina.boutoustous@univ-fcomte.fr (K. Boutoustous).

0957-4158/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mechatronics.2011.05.005

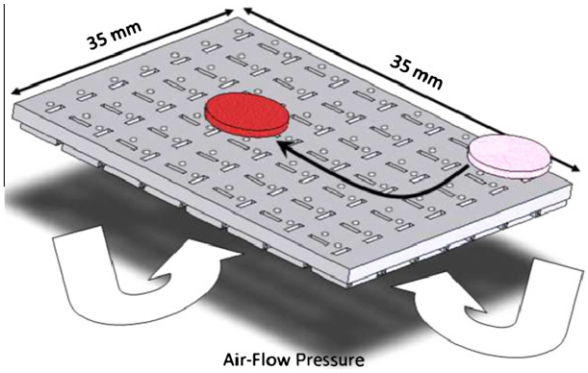

Fig. 1. General overview of the smart surface.

differentiation is presented in Section 4. Distributed part differentiation with gaps is proposed in Section 5. In Section 6, we present the 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. The following operations must then be performed on parts: identification, orienting, sorting, positioning and conveying. Among the most promising solutions to perform these tasks, is the combination of MEMS in order to form a microrobotics array.

There have been numerous projects on MEMS actuator arrays since the 1990's. Pioneering research works have developed different types of MEMS arrays 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 have failed to propose fully integrated solutions at a micro-scale (e.g. see [2]).

The goal of the Smart Surface project (see [13,14]) is to design a distributed and integrated micromanipulator based on an array of micromodules or cells and to develop an automated positioning and conveying surface. Each micromodule will be composed of a microactuator, a microsensor and a processing unit. The cooperation of cells thanks to an integrated network will permit the system to differentiate parts and to control microactuators in order to move and position accurately the parts on the smart surface. We consider tiny parts that cover a small number of cells and that are moved via air nozzles actuators. The actuators are represented by rectangular holes in Fig. 1. Air-flow comes through a microvalve 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 benefit of this solution is to ensure protection of the most fragile parts of the surface i.e. actuators. Note that circle holes correspond to microsensors in Fig. 1.

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 four neighbors (see Fig. 3).

We want to obtain in a distributed way a global knowledge of the discrete state of the smart surface, i.e. a discrete representation of parts that lay on the smart surface and their position. In [9], we have made the assumption that there is at most 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 subvectors $x_i \in \{0,1\}^n$, $i \in \{1,\ldots,n\}$, where x_i denotes the augmented local state of the i th 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 i-th cell 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. $x_{i,j}$, $j \in \{1,\ldots,n\}$, $j \neq i$. The mapping F permits one to obtain a mathematical formulation of the state acquisition problem and to derive several useful distributed algorithms.

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:

$$\begin{aligned}
&F_{i,i}(x) = x_{i,i}, \quad i \in \{1,\ldots,n\},\\
&F_{i,j}(x) = x_{i,j} + x_{j,j}, \quad \text{if } j \in N(i), \quad i \in \{1,\ldots,n\},\\
&F_{i,l}(x) = x_{i,l} + \sum_{j \in N(i)} x_{j,l}, \quad i,l \in \{1,\ldots,n\}, \quad l \notin N(i),
\end{aligned}$$

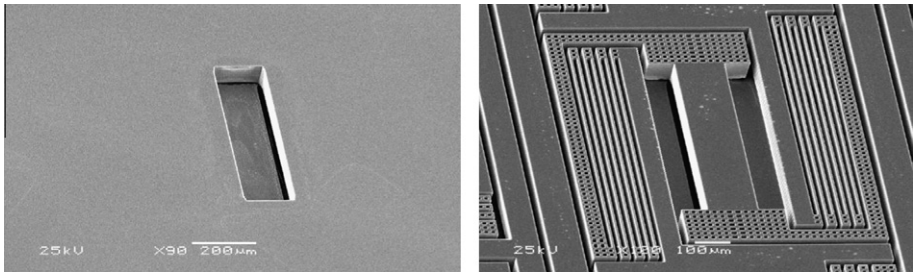

Fig. 2. Front-side and back-side of a microactuator.

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

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):

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

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

$$x^0_{i,j} = 0, \quad \text{if } j \neq i, \quad i,j \in \{1,\ldots,n\},$$
$$x^0_{i,i} = 1, \quad \text{if there is a part on cell } i, \quad i \in \{1,\ldots,n\},$$
$$x^0_{i,i} = 0, \quad \text{if there is no part on cell } i, \quad i \in \{1,\ldots,n\}.$$

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 $x', x' \geqslant x^*$, where the order relation is component wise, i.e. $x'_{i,j} \geqslant x^*_{i,j}$, $\forall i,j \in \{1,\ldots,n\}$. A simple illustration can be obtained by choosing $x \in E$ such that $x_{i,j} = 1$, $\forall i,\ j \in \{1,\ldots,n\}$. In this case, we clearly have: $x = F(x)$. However, the values of the component $x_{i,i} = 1, \forall i \in \{1,\ldots,n\}$ 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 $x^0 \leqslant x^*$ (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 is monotone.*

Proof. From the definition of the mapping F, we clearly have $F(x) \leqslant F(x')$, $\forall\ x, x' \in E$ *such that* $x \leqslant x'$; □

Theorem 2. *The distributed discrete iteration* $\{x^k\}$ *starting from* x^0 *defined as above converges to* x^*

Proof. From the definition of mapping F and x^0 we have

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

Moreover, we have

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

and

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

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

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

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

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

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; □

Let d' denote 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 message passing 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 a 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 all augmented local states are similar at convergence and that all cells have the same vision of the global state of the smart surface at most after $d' + 1$ iterations. □

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

```
Distributed synchronous discrete algorithm
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
```

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 any 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 $T = \{0,1,2,\ldots\}$ at which one or more sub vectors x_i, $i \in \{1,\ldots,n\}$, 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 $L_i = \{s_{1,i}(k),\ldots,s_{n,i}(k)\}$ be the subset of labels used during the updating phases of cell i with:

$$0 \leqslant s_{j,i}(k) \leqslant k, \quad \forall j,i \in \{1,\ldots,n\}, \forall k \in T(i).$$

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

We assume that $\lim_{k\to\infty} s_{j,i}(k) = +\infty, \forall j, i\{1,\ldots,n\}$, 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 $T(i)$, $i = \{1,\ldots,n\}$, 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 3.1.

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

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

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

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:

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

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. □

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

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

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 efficiency of this type of 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 the 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 permanent network faults or cell faults, due for example to sensor faults (giving a faulty local state) or processor faults (leading to update errors or no update release) a distributed asynchronous algorithm may end with an approximation of the solution that is different from x^*. However, we shall see in the next Section that the use of the gap-based techniques developed in this study may permit one to overcome this difficulty. Finally, we note that such an asynchronous algorithm has no deadlock.

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 a message, i.e. an update to its neighbors; this message activates the neighbors that are referred to as the 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 evolves according to the messages received and satisfaction of the local 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 conditions are satisfied and there is no message in transit in the network. This 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 th 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 the root R); they become active when receiving a message. The ith cell becomes inactive when the following asynchronous local stopping criterion is satisfied.

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

All cells can be activated many times, but the root R which is active once only. 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.

```
Distributed asynchronous discrete algorithm
For i from 1 to n do
  While Active (R) = True
    If Active (i) = True then
      k := k + 1
      x_i^k := F_i(x_1^{s_{1,i}(k)}, ..., x_n^{s_{n,i}(k)})
      If x_i^k ≠ 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 from n_i(j)
      End do
  End While
End do
```

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

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

4. A first approach to distributed part differentiation

We assume that there are a limited number of types of parts. In this paper, we shall consider that we have three types of parts. Thus, any part on the smart surface will belong to one class among these three classes. Nevertheless, the proposed algorithms work well when used with more classes.

This Section presents the first approach we have proposed for part differentiation in a low resolution context and under assumption that there is at most one part on the smart surface (see [3]). This approach is based on the computation of criteria. 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. The different criteria used in this paper are detailed in [3] (see also [22]).

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

- Offline stage: a database that contains the values of criteria used to differentiate parts is produced. Only a limited number of parts that will be called reference parts are considered. The values of the criteria are stored in cells.
- Online stage: cells try to differentiate the part on the smart surface by comparing the criteria values of this part with the values stored in the database.

4.1. Offline stage

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

- the reference part is rotated several times. Each time, the part turns around the center point and is also moved along s/10 cells, where s denotes the width of the part;
- a matrix of sensor values is generated, the matrix fits the smart surface, i.e. there is one entry per sensor, an entry is equal to 1 if the associated sensor is covered by a part, otherwise it is equal to 0;
- sub-matrices without rows and columns that contain only entries with value equal to zero are generated, these sub-matrices are the so-called masks, multiple copies of the same mask are discarded;
- values of the different criteria are calculated for all masks of the reference part.

The set of criteria values forms the database that will be used in 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 parts in real-time. Once cells of the smart surface have made acquisition of the binary representation of the part on the surface, they compute the criteria values. These values are compared to the criteria values in the database that are obtained at the offline stage. If there is a criterion value or a combination of criteria values that matches exactly one in the database, then the part is considered to be differentiated. If no such correspondence can be found, then no decision is taken.

5. Distributed part differentiation with gaps

In this Section, we propose an original distributed differentiation method based on gaps. As in the previous Section, we assume that types of parts are available in limited number. We recall that we consider in this paper the case with three reference parts that are also called registered parts. Nevertheless, our algorithm works well with more reference parts or in the case where several parts of different type lay on the surface.

We take benefit of the high level of parallelism available on the array of micromodules to derive a distributed algorithm. Cells 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 all cells that are covered by a part have differentiated the parts.

We detail now the decision process. We note that the value of the 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 a 3×3 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 surface (see Fig. 4 obtained with SSS, a multithreaded Java Smart Surface Simulator that will be detailed in the next Section). A view of an actual part on the smart surface and its discrete representation is also shown in Fig. 5.

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 registered part. Each cell compares measured values of the criteria for the current position of the part on the smart surface with values of criteria for a single reference position of the registered parts (those values are stored in the 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, e.g. sensor faults (giving a faulty local state) or processor faults (leading to erroneous discrete representation of a part).

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 magnify tiny differences between parts, like product of the differences between consecutive columns and consecutive lines are discarded. In the sequel, the number of considered criteria will be denoted by q. Let m denote the number of registered parts. Let $r_i(j)$ denote the reference value of ith criterion of the jth registered part (those values are computed offline). Let c_i denote the value of the ith criterion of a

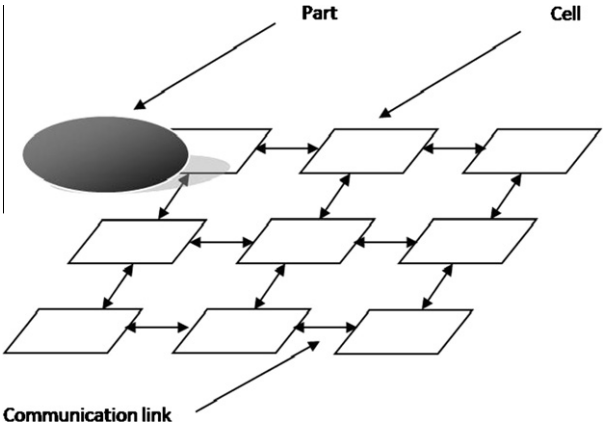

Fig. 3. Communication network of the smart surface.

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

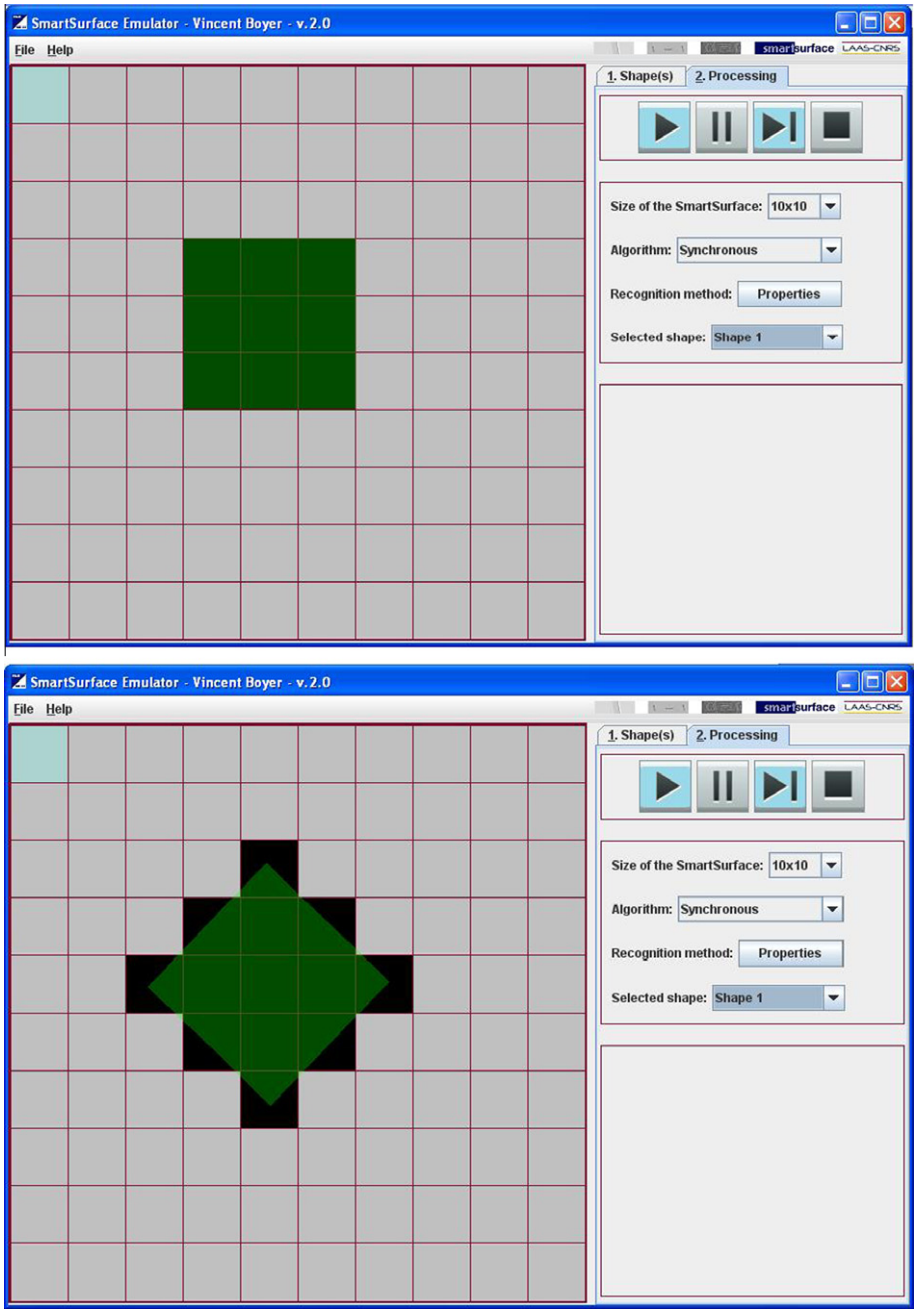

Fig. 4. Examples of different surface values for the same square with different orientations (SSS screenshots).

part on the smart surface. All cells compute the following gaps for the part that covers them:

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

The second approach relies on the use of a set of reference positions for the registered parts. We take into account rotations of parts. Without loss of generality and for symmetry reason, only rotations with one degree angle up to 45° are considered. Let $D = \{1,\ldots,45\}$ we denote by $r_i^d(j)$ the reference value of ith criterion of the jth registered part with d degrees angle; those values are computed offline. We denote by $C(j)$ the set of all reference values for jth part: $C(j) = \left\{\left(r_1^d(j),\ldots,r_q^d(j)\right), d \in D\right\}$. All cells compute the following gaps for the part that covers them:

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

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

Decision making concerning part differentiation at each cell relies on the respective values of the gaps. The part j that is chosen corresponds to the gap $g(j)$ or $g'(j)$ that is the closest to zero. We note that cells make computations concurrently.

Concurrent decision making based on gaps is particularly interesting with respect to parts positioned any manner on the smart surface or faults that may occur on the smart surface, e.g. sensor failures. The use of gaps is also interesting with respect

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

to differentiation of parts that are slightly altered, i.e. parts that are slightly modified.

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 part differentiation 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.

The simulator permits one to build a smart surface with any size and different basic parts like squares, rectangles, parts with L shape or I shape and so on, that will become reference parts or given part on the surface. SSS permits also one to place the generated part everywhere on the smart surface. It is possible to rotate shapes on the smart surface and to introduce sensors faults (see Fig. 4, for a square shape). The simulator allows one to choose and carry out a synchronous or asynchronous distributed state acquisition algorithm and to display dynamically the augmented local state of any cell (see right window of Fig. 6 that corresponds to iteration 2). One can have a dynamic view at the activity graph of the smart surface, i.e. one can see the cells that are active (the cells who are updating their augmented local state). One can also choose particular criteria and a differentiation method, e.g. gaps based methods or total differentiation methods studied in [22]. Finally, one can display the results of the part differentiation phase for the different criteria selected; SSS permits also one to display some statistics.

In SSS, each cell is managed by a thread. Cells communicate with their neighbors via buffers (see Figs. 7 and 8). 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.

Memory organization is displayed in Fig. 7. For a typical cell with four neighbors, data sent by neighbors, according to the communication scheme shown in Fig. 8, are stored in dedicated buffers, e.g. Top Buffer, for neighbor that is above the considered cell and Right Buffer for neighbor that is situated on the right side of the cell. Finally, the so-called Local State is used to store the current value of the augmented local state of the cell. This organization presents great flexibility in order to carry out different state acquisition algorithms. In particular, it permits one to implement easily asynchronous algorithms since neighbors can freely exchange data with a cell without synchronization. Nevertheless, it permits also one to implement synchronous algorithms whereby neighbors must be granted permission to access buffers.

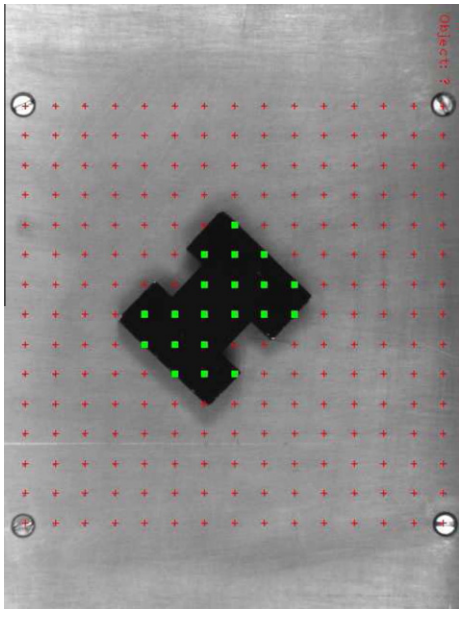

Fig. 5. View of an actual H shape part on the smart surface and its discrete representation (courtesy of Femto-ST).

7. Tests

The multithreaded Java Smart Surface Simulator has been carried out in parallel on a multi-core machine with 3.0 GHz 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 referred to as the Sq part, an L shape part and I shape part. All parts have been randomly placed on the surface 200 times, leading to 200 draws with SSS. 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 have been selected.

Table 1 displays the results with the top 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 permitted us to correctly differentiate the part Sq in 37% of cases.

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 with criteria combination. We note that the part Sq was correctly differentiated in 59% of the cases; which is better than with criterion *S* or criterion *A* alone. The average differentiation rate was increased from 40.83% (for criterion *A* alone) and 48.76% (for criterion *S* alone) up to 67.16% for the combination of *A* and *S*.

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*.

Table 4 gives the results obtained with the gaps *g*′.

These results show that the methods based on gaps *g* and *g*′ give better differentiation rates than the total differentiation method in the following cases:

- With a single criterion, e.g. the criterion S, the average differentiation rate can reach 99% with *g* (see first row of Table 3) and 99% with *g*′ (see first row of Table 4); we recall that the average differentiation rate is 48.76% with the total differentiation method (see first row of Table 1).
- With a combination of criteria, the use of methods based on gaps permits also one to improve the average differentiation rate. We note that the average rates are 98.83% (see third row of Table 3) and 92.83% (see third row of Table 4), respectively, for g and g', respectively. We recall that the average differentiation rate is 67.16% with the total differentiation method (see first row of Table 2).

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, in some cases one criterion alone may be sufficient to reach almost 100% differentiation rate.

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

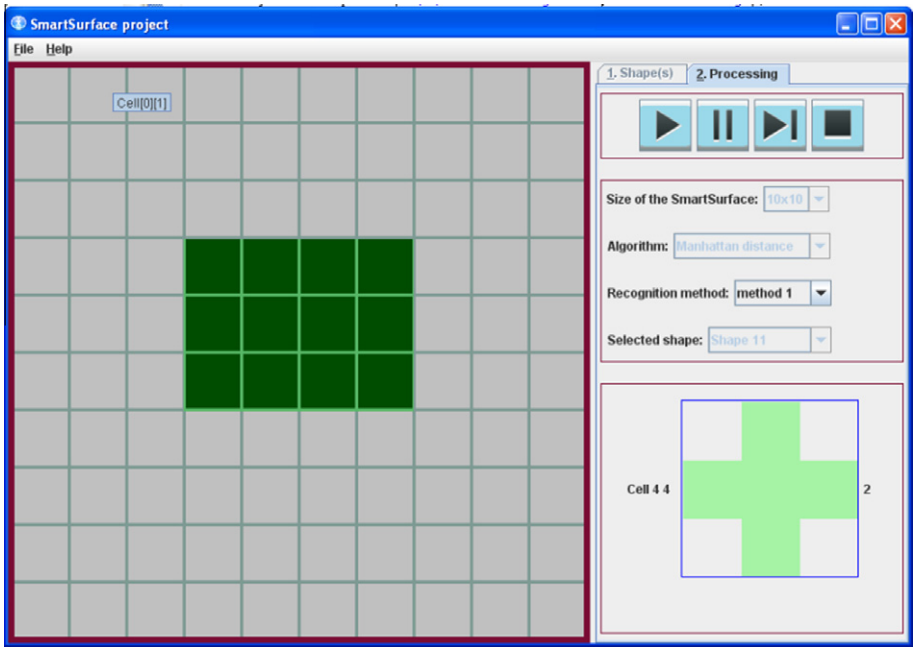

Fig. 6. SSS smart surface window (left) and "extended" local state window of a given cell at iteration 2 (right).

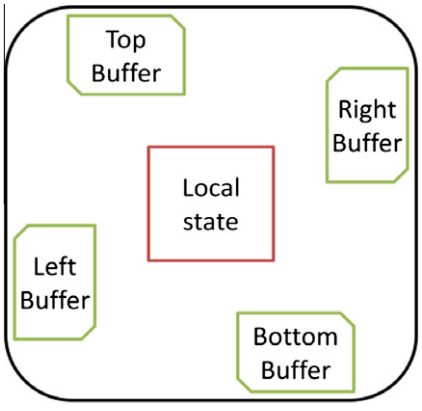

Fig. 7. Basic cell architecture with SSS.

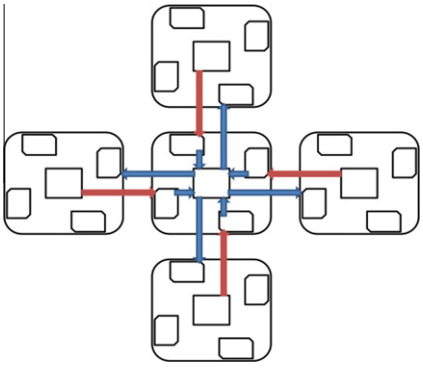

Fig. 8. Cell communication scheme with SSS.

Table 1
Differentiation rates for the criteria a and s,total differentiation.

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

Table 2
Differentiation rates for criteria combination,total differentiation.

Criteria	Sq (%)	*I* (%)	*L* (%)	Average
A and S	59.50%	74.00%	68.00%	67.16%

Table 3
Differentiation rates with the first gap.

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 4
Differentiation rates with the second gap.

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

8. Conclusions and future work

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

The approach developed in this paper is particularly useful to MEMS related applications in semiconductors, micromechanics

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005

and pharmaceutical industry. The cooperation of integrated micro-modules forming an array and the design and analysis of distributed solutions will facilitate the development of fault tolerant industrial applications whereby microparts or microitems will be sorted, conveyed or positioned precisely.

Actual implementation of the proposed techniques on an industrial or experimental distributed smart surface platform has to be made in order to complete the study.

We shall concentrate on concurrent part differentiation methods that exploit smart surface natural parallelism, e.g. each micro-module i can apply a mapping T_i to extended local state x_i and compute some criteria. Typically, mappings T_i can be some rotations. This kind of preconditioning can enrich or speed up part differentiation. It may also be interesting to derive part differentiation techniques that are not criteria based and which exploit directly the part code.

We plan to study combined part differentiation and part motion; it may be efficient to differentiate parts while moving them on the smart surface. The reader is referred to Ref. [23] for a first study.

Finally, future directions of research will concern fault detection and solutions that propose degraded but everlasting behavior. MEMS-based reconfigurable solutions for sorting conveying or positioning microparts will be investigated.

Acknowledgements

The authors thank Guillaume Laurent and Nadine Le Fort-Piat, Femto-ST, Besançon France, for photos of the parts and information regarding the smart surface.

References

[1] Biegelsen D, et al. Airjet paper mover. In: SPIE international symposium on micro machining and micro fabrication, 4176-11, September 2000.
[2] Fukuta Y, Chapuis Y, Mita Y, Fujita H. Design fabrication and control of MEMS-based actuator arrays for air-flow distributed micromanipulation. IEEE J Micro–Electro-Mech Syst 2006;15(4):912–26.
[3] Boutoustous K, Dedu E, Bourgeois J. A framework to calibrate a MEMS sensor network. In: Proceedings of the International Conference on Ubiquitous Intelligence and Computing, Brisbane Australia, July 2009. Lecture Notes in Computer Science. Berlin: Springer; 2009. p. 136–49.
[4] Smart Surface ANR project; 2010. <http://www.smartsurface.cnrs.fr/>.
[5] Smart Surface project at LAAS-CNRS, Toulouse; 2010. <http://spiderman-2.laas.fr/SMART-SURFACE/>.
[6] Ishida H, et al. Recognition of low resolution characters by a generative learning method. In: Proceedings of the 1st international workshop on camera-based document analysis and recognition; 2005. p. 45–51.
[7] Tabbone S, Wendling L, Salmon J-P. A new shape descriptor defined on the Radon transform. Comput Vis Image Understand 2006;102(1):42–51.
[8] Kare J. Backyard star wars, IEEE Spectrum, May 2010. p. 26–9.
[9] El Baz D, Boyer V, Bourgeois J, Dedu E, Boutoustous K. 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] Fujita H. Group work of microactuators. In: International advanced robot program workshop on micromachine technologies and systems, Tokyo, Japan, October 1993. p. 24–31.
[11] Pister KSJ, Fearing R, Howe RT. A planar air levitated electrostatic actuator system. In: IEEE Micro electro mechanical systems. An investigation of micro structures, sensors, actuators, machines and robots; 1990. p. 61–71.
[12] Bohringer KF, Bhatt V, Donald BR, Goldberg KY. Algorithms for sensorless manipulation using a vibrating surface. Algorithmica 2000;26(3-4):389–429.
[13] Le Fort-Piat N, et al. Smart surface based on autonomous distributed micro robotic systems for robust and adaptive micromanipulation, ANR Smart Surface Project Proposal; 2006.
[14] Matignon L, Laurent G, Le Fort-Piat N. Design of semi-decentralized control laws for distributed-air-jet micromanipulators by reinforcement learning. IROS 2009:3277–83.
[15] Robert F. Discrete iterations, a metric study, Springer series in computational mathematics. Berlin: Springer-Verlag; 1986.
[16] Bertsekas D, Tsitsiklis J. Parallel and distributed iterative algorithms: a selective survey. Automatica 1991;25:3–21.
[17] Radid A. Itérations Booléennes et sur des ensembles de cardinal fini, Ph.D. of the Franche-Comté university; 2000.
[18] Dijkstra EW, Scholten CS. Termination detection for diffusing computation. Inform Process Lett 1980;11:1–4.
[19] Bertsekas D, Tsitsiklis J. Parallel and distributed computation: numerical methods. Athena Scientific; 1997.
[20] El Baz D. An efficient termination method for asynchronous iterative algorithms on message passing architecture. In: Proceedings of the international conference on parallel and distributed computing systems, Dijon, vol. 1; 1996. p. 1–7.
[21] El Baz D. A method of terminating asynchronous iterative algorithms on message passing systems. Parallel Algorithms Appl 1996;9:153–8.
[22] Boutoustous K, Dedu E, Bourgeois J. An exhaustive comparison framework for distributed shape differentiation in a MEMS sensor actuator array. In: Proceedings of the international symposium on parallel and distributed computing (ISPDC). Kraków, Poland: IEEE Computer Society Press; 2008. p. 429–33.
[23] Boutoustous K, et al. K. Distributed control architecture for smart surfaces. In: Luo RC, Asaman H, editors. IROS 2010, 23-rd IEEE/RSJ international conference on intelligent robots and systems. Taipei, Taiwan: IEEE Computer Society Press; 2010. p. 2018–24.

Please cite this article in press as: El Baz D et al. Distributed part differentiation in a smart surface. Mechatronics (2011), doi:10.1016/j.mechatronics.2011.05.005