Tree-structured knowledge in a distributed intelligent MEMS application

Atsushi Sato, Eugen Dedu, Julien Bourgeois, Runhe Huang

Tree-structured knowledge in a distributed intelligent MEMS application

Atsushi Sato, Eugen Dedu, Julien Bourgeois and Runhe Huang

***Abstract*—This paper proposes the tree-structured knowledge approach for performing part recognition in controlling MEMS-arrayed manipulation surfaces. In this approach, a new data structure, a tree-structured array, is used to store knowledge about models of the objects at an offline stage and to accumulate and share knowledge among neighboring active cells about shapes of objects which must be reconstructed and differentiated on a MEMS-arrayed surface at the online stage. Comparing this approach with the previous matrix-based approach, which contained redundant information in each cell and communication, and demanded excessively frequent comparison in shape differentiation, the current tree-structured knowledge approach aims to use one model for a shape in database, reducing the memory footprint, and avoiding frequent comparison in the differentiation phase. In this paper, both approaches are analysed and compared. Though the current approach shows better performance in terms of a smaller memory footprint and lower communication cost, it trades off the reduction of memory footprint against the probability of the differentiating.**

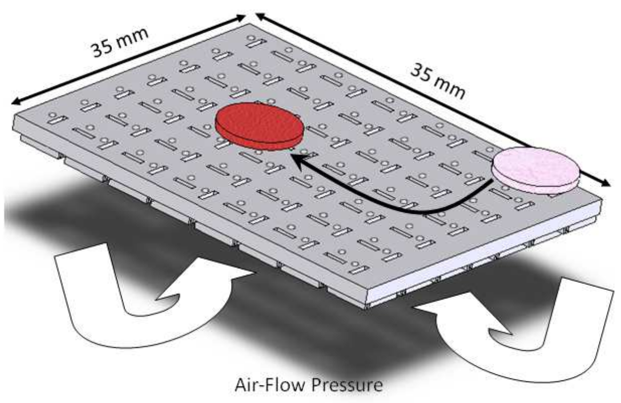


Figure 1. An overview of the Smart Surface [13].

I. Introduction

Micro-Electro-Mechanical Systems (MEMS) are now a mature field of research as well as a promising technology ready for mass-production. Examples of successful products can be found in diverses area like accelerometers, inertial measurement units (IMU, that are now included in airbag systems as well as in most of the recent smartphones or laptops), bubble ejection systems of inkjet printers and digital micromirror devices (DMD, technology used for projection displays). If the first applications use single MEMS, the latter uses distributed MEMS organized in an array.

Our current research objective is to include sensors and to add intelligence to distributed MEMS in order to build what we call distributed intelligent MEMS (diMEMS). One application of diMEMS is a pneumatic distributed conveyor called *Smart Surface*, for conveying, fine positioning and sorting of very small parts (see Figure 1). Before being conveyed to the right position, the parts should be first recognized.

Distributed manipulation has been a very active topic since the 1990's. These pioneer researches have developed different types of distributed manipulators, based on servoed roller wheels [1], cilia actuators [2], [3], [4], [5], [6], suction nozzles with air-hockey tables [7], [8], [9], [10] and directed air-jets [11], [12], [13]. All these works require a centralized control and are, therefore, not scalable.

A. Sato and R. Huang are with Hosei University, Japan, atsushi.sato.9q@stu.hosei.ac.jp(A. Sato), rhuang@hosei.ac.jp(R. Huang)

E. Dedu and J. Bourgeois are with UFC/FEMTO-ST, France, {Firstname.Lastname}@femto-st.fr

Some of these preliminary studies use sensorless manipulation schemes based on Goldberg's algorithm [14] for parallel jaw grippers. The jaw grippers are obtained with actuator arrays by creating opposite field forces which then can orient and move the parts. Böhringer *et al.* [15] have proposed a programmable vector field method based on Goldberg's algorithm. This sensorless scheme is well-adapted for contact manipulation arrays but has shown some limitations when applied to contactless manipulators [16]. For instance, the absence of closed-loop control has led to undamped oscillations and unwanted behaviors.

As referred previously, the Smart Surface have to take into account functions such as recognition, conveyance and positioning of an object. The implementation of these functions must also meet the requirement of scalability, modularity and robustness of distributed manipulation systems. Moreover, since the objects are small compared to sensors and can be rotated, classical recognition methods such as neural networks are not appropriate.

We consider that the global system can be composed by a large number of cells, each cell being able:

- to sense locally the state of the object (for example the presence/absence of the object);
- to act locally on the object;
- to decide its action by itself.
- to communicate with its 4-neighbors (von Neumann neighborhood)

Figure 1 illustrates the Smart Surface concept. Each cell receives measures from its sensor, acts on the object by its actuator and is able to communicate with its four neighbors.

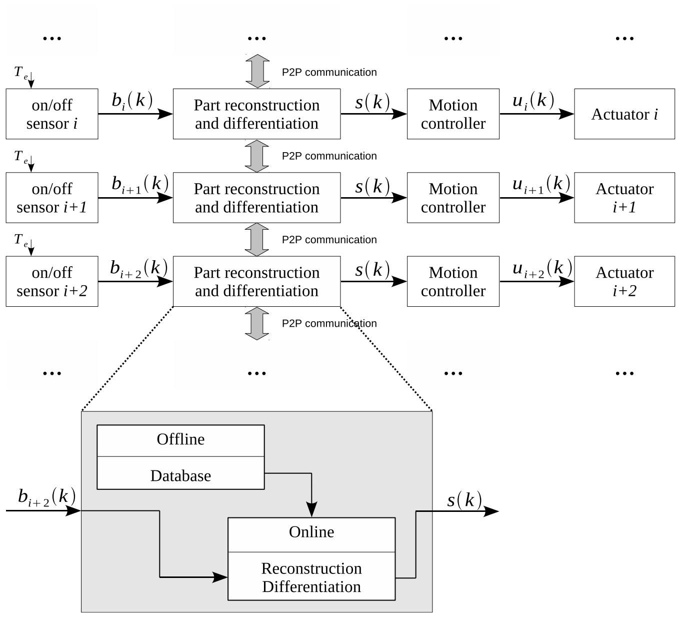


Figure 2. The distributed control architecture.

A programmable processing unit is able to perform recognition tasks and calculation of local control laws.

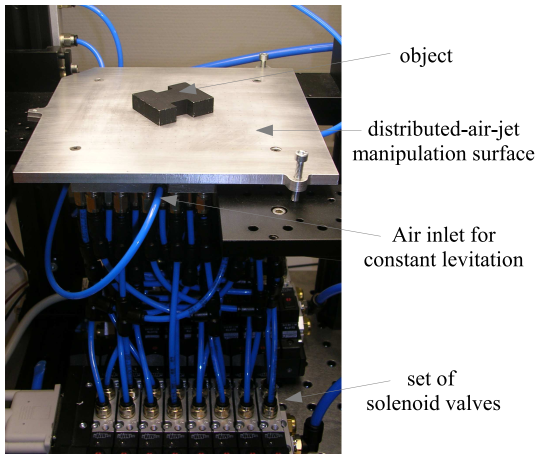
The distributed control architecture is able to perform part recognition and close-loop control of the Smart Surface. This architecture is composed of an array of cells where each cell is linked to a local sensor and to an actuator. A communication network allows each cell to exchange messages with its 4-direct neighbors. Using the adjacent network, the cells work autonomously and in a distributed manner by exchanging their local information of the object.

Figure 2 details the distributed control architecture. The state $b_i(k)$ of the sensors is received by part-reconstruction modules. These blocks communicate with their neighbors in order to reconstruct the image of the object. Then, the differentiation functions use the reconstructed image to recognize the shape of the object and to give the proper information $s(k)$ to the controller blocks. Then, the motion controllers independently allocate a state $u_i(k)$ to each actuator i so as to generate a specific motion of the object. The functions of reconstruction, differentiation and control are implemented following a fully distributed scheme.

In the past we used as distributed-air-jet manipulation surface a $120 \times 120\ mm^2$ square surface upon which an object is moving in aerodynamic levitation (see Figure 3) [17].

In [17], we presented a differentiation algorithm which used criterion values to store shapes. We think this differentiation algorithm can be improved by using an entity intelligence pool and its sharing among objects [18].

The current paper presents a new data structure, a tree-structured array used to store information about models of the objects which must be differentiated. Additionally, the data exchanged uses the same tree-structured array and they are interestingly used to differentiate objects. This allows a better memory footprint and a better way to represent knowledge.



object
distributed-air-jet manipulation surface
Air inlet for constant levitation
set of solenoid valves

Figure 3. The distributed-air-jet manipulation surface.

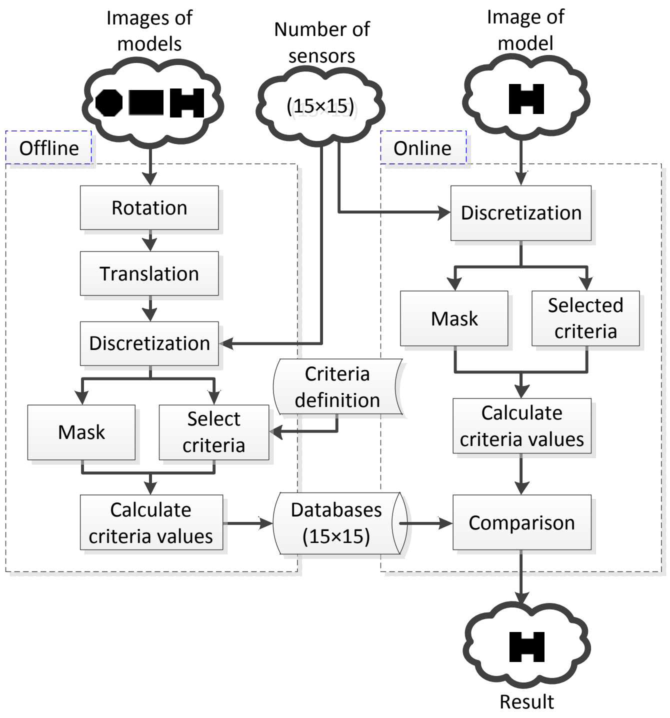


Figure 4. The process of previous approach simulation.

II. Object differentiation overview

Given a set of objects, the Smart Surface will have to differentiate all the parts within the set. As the processing power of the Smart Surface is embedded in very limited space, this differentiation process has to be optimized both in term of memory used and processing power needed. Figure 4 illustrates the 2-stages differentiation process used in a previous approach.

The offline stage is executed before using the real device i.e. the Smart Surface. It aims to construct a database in which each model (a model is an object which can be placed on the Smart Surface) is characterized by a set of criteria values, such as surface and perimeter, as defined in [19]. In a first step, the images of models are discretized in matrices by affecting 1 if the sensor is covered by the object and 0 otherwise. In a second step, images are rotated from $1°$ to $359°$ in $1°$ step. Then, masks are generated from the matrices and duplicates are removed. Finally, the set of criteria [19] used for the differentiation is chosen. Their values are calculated from the masks and are stored in the database; the database is uploaded in every cell.

The online stage has two phases: reconstruction of the object in each cell, and then differentiation of the object.

a) Reconstruction phase: Once the object is put on top of the Smart Surface, each cell knows its status (covered or not by the object), but it doesn't know the status of its neighbors. The cells covered by the object are called *active cells*. Each active cell does an iterative process to reconstruct the binary representation of the object. This iterative process consists in two steps: communication and computation steps.

In the communication step, all active cells communicate with their neighbors. This communication is done by sending a message, which consists of a matrix of bits, the matrix size is the same as the smart surface size (1 bit per cell). Throughout the re-iteration of the communication phase, all the active cells will extend their state thanks to the received matrices from their neighbors, until they reach a proper representation of the object.

The computation step consists in updating the view of each cell according to the views collected from its neighbors in the communication step. This update consists in applying the union operator ($\cup$) bit by bit between all received matrices and its current one.

It can be proved [20] that after N steps (N is the height plus the width of the object), this algorithm converges and each cell has the binary representation of the object currently on the Smart Surface. The number of steps is therefore not dependent on the Smart Surface size.

b) Differentiation phase: Once each cell has the binary representation of the object, the criteria are computed. These values are compared to the criteria values for each model in the database, computed in the offline stage. The differentiation is successful if there is a criterion or a combination of criteria which uniquely identifies the object.

III. The Tree-Structured Knowledge (TSK) based alternative approach for P2P communication

A. Background

Rapid advances in electronics, electro-mechanics, and nanotechnology, have lead to ubiquitous devices [21] called u-objects [22], [23] like mobile devices, sensor devices, RFID devices, wireless devices [24], [25], wireless wearable devices, etc. are getting ever more common and ever smaller, even invisible to the naked eye [26]. This implies that u-objects must necessarily be of autonomic/smart/intelligent computing/processing ability in order to provide pervasive autonomic support/services be able to do right thing, at the right time, in the right place, and by the right way) for and instead of human users [27].

This removal of u-objects from the human scale also means their processing power and storage capacity resources are limited and constrained. Embedded Intelligence Entities (IE) (data + knowledge + reasoning engine) [18] in u-objects may be one theoretical solution but at the present time are impractical. However, Intelligence Entity Sharing (IES) from external systems is a practical alternative approach. Whenever necessary, a u-object can request/rent/borrow an intelligence entity from an intelligence entity pool (IEP) to provide intelligent/smart/autonomic support or services. The intelligence entity pool has to have an appropriate knowledge representing structure, with which the intelligence entity can be flexibly composed and decomposed.

One appropriate structure to represent knowledge is a tree-structure. A tree structure consists of a number of nodes and directed links. A directed link connects an upper level parent node and a lower level child node to represent the relation between them, and an arrangement of nodes and links allows a tree to represent knowledge efficiently. Sub-trees of a parent node. can even be cut off or attached to another node to actually form the processes of knowledge composition and decomposition. Knowledge represented in the tree-structure can be relatively easily constructed and the tree-structured knowledge base makes computational knowledge visually comprehensible. In this section, we discuss the implementation of the concept by applying tree-structured knowledge (TSK) to the Smart Surface.

The previous approach, in which matrices representing the state of sensors are sent as messages in P2P communications, was explained in Section II of this paper. The amount of data communication traffic is fixed and the updating method is simple. The matrix, however, has redundant information. The differentiation phase is repeated every time the matrix is updated, since the matrix does not have the trigger for the comparison with the models in the database.

In the current article, we propose a new online phase. It uses a tree-structured array as message data. The array has a variable length and each value gives the state of the sensor at a relative position: north, west, south, and east. Moreover, the values of child nodes are inserted to the right of the parent.

B. Reconstruction phase

The root of a tree is the state of the cell itself, and its child nodes are the trees of its four neighbors. The state of a cell is active when an object is above its sensor, or passive when there is no object-above.

The tree generation process, leading to the reconstruction of the current object image, takes the following steps:

1) Each active cell generates its initial tree-structured array which is only composed of its current state.
2) Each active cell sends to its four neighbors its tree, removing the values corresponding to the destination (see Figure 5)
3) Each cell receives trees from its four neighbors.

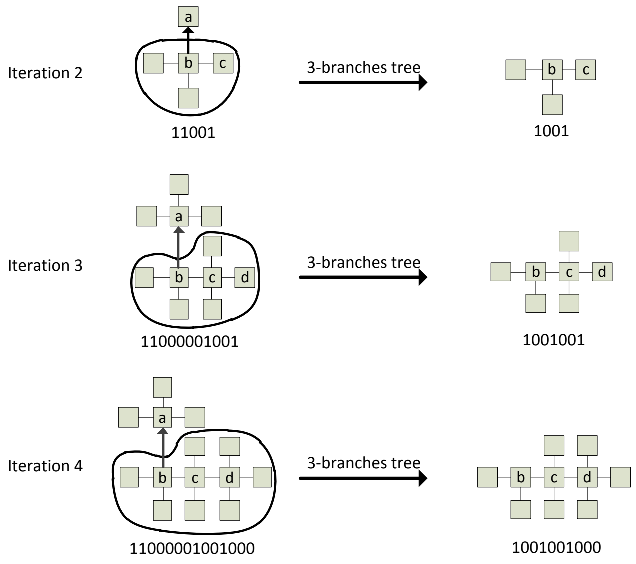


Figure 5. The tree-structured array of cell "b" and the three subtrees sending to cell "a" in the object reconstruction process.

4) Each cell updates its tree by replacing its four children with the four trees received.
5) If the leaf values of the received trees are all 0 or the trees do not contain any updates as no sensors have changed state, the reconstruction phase is over. If not the reconstruction algorithm continues with step 2.

As a result, the tree of each cell expands until the whole object is reconstructed.

As several different paths exist from one cell to another, a cycle can be created leading to an infinite growth of the trees. Such cycles are avoided through an algorithm briefly explained as follows. Since trees can be written as (tree-structured) arrays, replacing sub-trees is equivalent to connecting the current array with the array received. So, after connecting two arrays, a check for duplication of values of 1 is performed on the arrays received from neighbors (see Figure 6): If two 1s have the same coordinate, the latter (including its child nodes) is replaced with 0. The check is performed from left to right in arrays removing the right-most duplicate. An example as shown in Figure 6, the current array 10011 in the iteration 2 is updated by connecting two received arrays (1001 and 1010) from its two neighbors. The values, 1s , in the updated array and their associated coordinates are checked and the third and fifth 1 have the same coordination (1, 1), then the fifth is replaced with 0.

The reconstruction process is summarized as follows:

```
1:  {Reconstruction phase}
2:  Initialize its tree-structured array (create the root of the tree)
3:  repeat
4:      Generate the four tree-structured arrays for each neighbor
5:      Send its array to the neighbors
6:      Receive the arrays from its four neighbors
7:      Replace its four children with the received arrays
8:      Check duplication of the value, 1
9:      Replace the duplicate values with 0
10: until Receive the array the leaf values of which are all 0 or the
    unchanged array values from its neighbors
11: {Differentiation phase (presented later)}
```

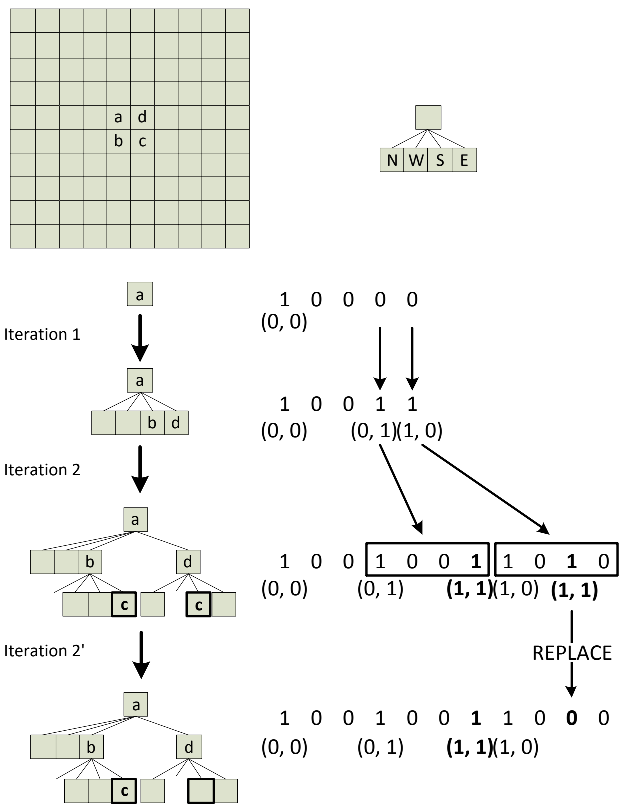


Figure 6. Cycle avoidance during object reconstruction process.

Figure 7 shows a complete example of the reconstruction process of an object. In this figure, each square is a cell in the Smart Surface and every cell has four neighboring cells denoted as "N", "W", "S", and "E" correspond to their relative positions. In this example, there are four active cells, named "a", "b", "c", "d" and each cell receives the state of the other cells in four communications. Every cell has four neighbors, but one of them as the destination node which corresponds to the parent node and the rest three are regarded as children nodes. Therefore, all nodes except the root have three children, and a line without square represents the position of the parent node.

C. *Differentiation phase*

To differentiate between objects, a database of storing shape models is required. This database is constructed during an offline stage. In the previous approach, a number of criteria corresponding to the shape of an object on the Smart Surface are generated and stored in the model database. Instead, in the current approach, each shape model is stored as one binary array which corresponds to a 4-branch tree. To simplify the differentiation process, the root cells of all the arrays in the database have to be fixed as far north as possible, then as far west as possible. The array generated in the reconstruction phase therefore has to be transformed into a unique array

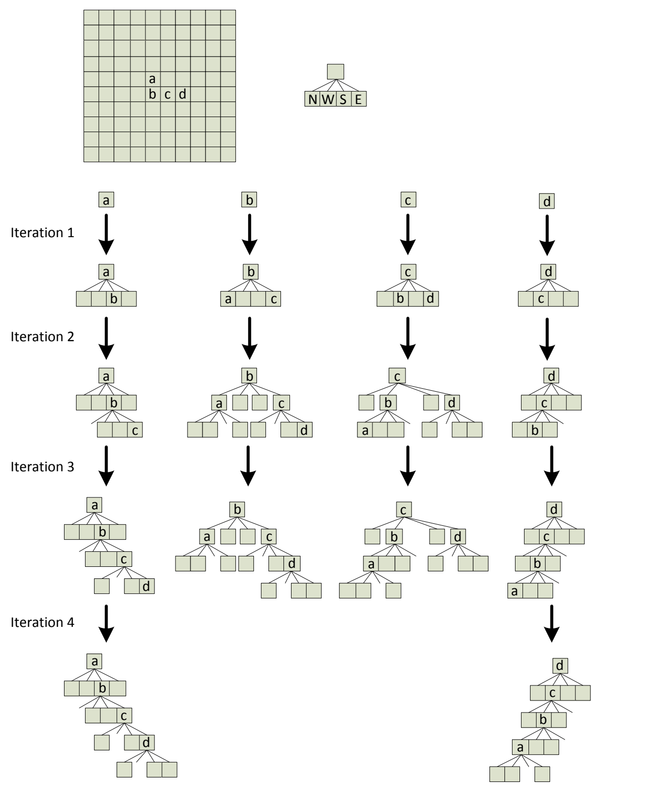


Figure 7. The tree evolution in each active cell during object reconstruction process.

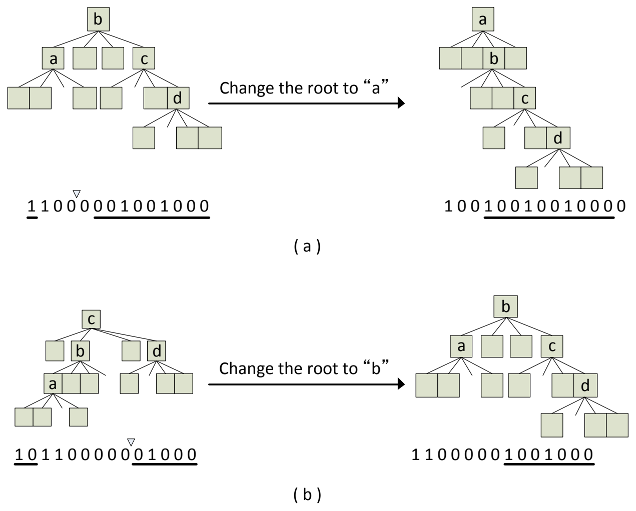


Figure 8. Array transformation to change the root cell.

the root cell of which is the most northern and western cell. However, at the online stage, each cell is of a tree-structured array and it is evolving in each iteration step. For non-root cells, the transformation from a non-root cell's array to a root cell's array is performed. At first, it is to scan the tree-structured array from the most left. Since the first bit is always the state value of the cell itself, it checks the second bit. If its value is 1, it means the cell has an active north neighboring cell, and then the north neighboring cell is changed to be the root cell. As illustrated in Figure 8(a), the north neighboring cell "a" is changed to be the root cell, instead of cell "b" by

1) adding 1 to the head of the sub-array (001001000) of cell "b", and
2) inserting them (1001001000) to the position (pointed by an arrow) between the 4th and 5th bits, and finally
3) deleting the first bit (1) and the resulting array is 10010010010000 from the original array, 11000001001000.

If the second bit's value is 0, it means no active cell is found at north. Similarly, the transformation algorithm checks if the cell has an active west neighboring cell. If it has, the west neighboring cell is changed to the root cell in the similar steps. As shown in Figure 8(b), cell "b" is changed to be the root cell instead of cell "c" as following steps:

1) adding 10 to the head of the sub-array (01000) of cell "c", and
2) inserting them (1001000) to the position (pointed by an arrow) between 9th and 10th bits, and finally
3) deleting the first two bits (10) and the resulting array is 11000001001000 from the original array, 10110000001000.

As described above, each array is checked to ensure the root node of its associated tree is the most northern and western root cell. For a complicated shape differentiation, it is possible that some of the arrays may not be able to be correctly transformed such that the associated cells may not be able to successfully differentiate the shape of an object, namely, no matching with a shape model in the database. In such case, it has to wait for other cells which can differentiate the shape of an object in database. Once the shape model is differentiated, the differentiation result and the position of the object (coordinates of the bounding rectangle) on the Smart Surface. The control block takes the responsibility from now on, either to give the object a small movement in order to retry differentiation or to move the object to its destination. This phase is summarized as follows:

```
1:  {differentiation phase}
2:  **repeat**
3:      **if** the north cell is active **then**
4:          transform the array to change the root to the north cell
5:      **else if** the west cell is active **then**
6:          transform the array to change the root to the west cell
7:      **end if**
8:  **until** there is no active neighboring cells on the north and west
9:  **if** the root is the most northern and western cell **then**
10:     **repeat**
11:         compare the array with all the models
12:         **if** discover the same array **then**
13:             send the differentiation result to the motion controller
14:         **end if**
15:     **until** the array is discovered or all shapes have been checked
16: **end if**
17: wait for the object to move to another place and restart the
    reconstruction phase
```

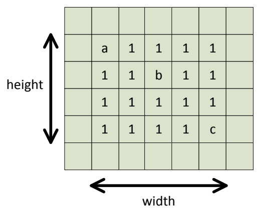


Figure 9. A shape of the rectangle model.

IV. Results and performance analysis

We evaluated the proposed approach in terms of the number of communication iterations, the amount of communication traffic, the length of computation time and the size of the memory footprint, and compared it with the previous approach presented in [17].

A. Number of communication iterations

The number of communication iterations of a cell depends on its position inside the shape. The edge cells need many communication to attain the opposite side cells, whereas central cells need fewer communications. The maximum number of communications is the sum of the height and width of the object minus one, while the minimum number is one plus half the sum of the height and width, rounding down to the nearest integer. For example, the height and width are four and five in Figure 9. Cell “a” needs to communicate eight times, but cell “b” only needs to communicate five times.

In the previous approach, the number of iterations is the same for both cells.

B. Communication traffic

As an example, the amount of communications in a 10×10 Smart Surface is calculated. The traffic amount is variable in the proposed approach, because the message consists of the states of cells known by the sender and the number of those cells increases with time. The amount of traffic for the example in Figure 7 is as follows. In the first step, all the messages have 1 bit. In the next step, the active cells send 4 bits, because the messages contain the states of their three neighbors (without the destination node). After that we show only the amount of the messages from cell “b” to cell “a”, because this amount depends on the sender and receiver. Cell “b” generates the sent message for cell “a” as shown in Figure 5. The amounts are 4 bits, 7 bits, and 10 bits, from top to bottom. This amount needs to be multiplied by four, since trees are sent to the four neighbors. The results for the other nodes are obtained in a similar way. Summing up, the amount of data communication traffic in each active cell is $(1 + 4 + 7 + 10) \times 4 = 88$ bits, and the amount of the traffic in the network is $88 \times 4 = 352$ bits (there are 4 active cells).

In the previous approach, 100 bits of data are sent between cells for communication. Since active cells communicate with their four neighbors, each active cell sends 400 bits of data at a time. If there are four active cells as shown in Figure 7, the cells need to communicate 4 times in order to get all active values, i.e. the amount of the traffic is $4 \times 4 \times 400 = 6400$ bits, about eighteen times greater than the proposed approach.

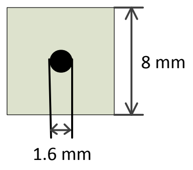


Figure 10. The size of a cell.

C. Computation time

The length of computation time consists of three main times: reconstruction time (TR), transformation time (TT) and comparison time (TC). Since they are iterated, we have taken into account the number of the iterations. The number of iterations for TR was calculated above. TT is the distance from the cell to the most northern and western cell except in cases where the root cannot be changed to the most northern and western cell. TC is the number of executions of the procedure (the reconstruction and differentiation phase) until the differentiation is successful, because the differentiation phase is performed after the completion of the reconstruction phase. As a consequence, the computation time is calculated as below:

$$T = \sum_{i=1}^{N_{Reconstruction}} TR_i + \sum_{j=1}^{N_{Transform}} TT_j + \sum_{k=1}^{N_{Comparison}} TC_k \quad (1)$$

where N_X is the number of iterations for X.

We consider the most extreme case in differentiating the object. This corresponds to cell “c” in Figure 9, as it is furthest from the northernmost and westernmost point. The number of communications and transformations are 8 and 7 respectively, and the number of comparisons is at most the number of shapes in the database.

To arrive at a number of shapes for this example, we posited the following conditions. The size of the Smart Surface is a 600×600 pixels square surface with a 15×15 MEMS array, cf. [28]. The size of a cell and MEMS are shown in Figure 10 and the object models are shown in Figure 11, taken from [17]. The criteria considered for differentiation are A, P and S, defined as follows:

- A (number of angles): The number of 1 having at least three neighbors to 0 and forming a right angle.
- P (perimeter): The number of sides having a neighbor at 0.
- S (surface): The number of 1 of the object.

The simulation of the offline stage of the three objects leads to 48 shapes for the circle, 248 shapes for the rectangle and 428 shapes for the “H” object. Assuming that all the 724 shapes are in the database, and that the differentiation is successful with 0 failures (since the rotation degree is 1°), and that all the coordinates for the position of objects are included with those in the offline stage, the shape of the object is found

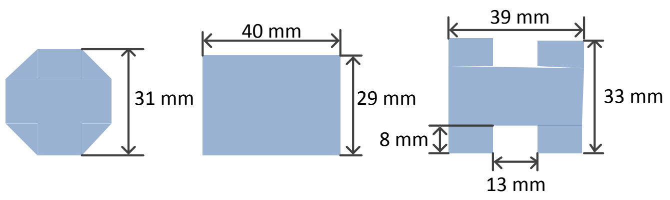


Figure 11. The size of the models.

necessarily after at most 724 comparisons, so the computation time becomes:

$$T = 8TR + 7TT + 724TC \tag{2}$$

On the other hand, the time with for the previous approach involves reconstruction time (TR'), criteria calculation time (TA) and comparison time (TC'):

$$T' = \sum_{i=1}^{N'_{Reconstruction}} (TR'_i + TA_i + \sum_{j=1}^{N'_{Comparison}} TC'_{ij}) \tag{3}$$

where $N'_{Reconstruction}$ is the number of communications and $N'_{Comparison}$ is the number of comparisons.

To calculate the time to differentiate the object for cell "c" (the most extreme case) in Figure 9, the number of communications is 8, and the number of comparisons is the same as the proposed solution. On the assumption that there are 50 criteria in the database, and that the differentiation is successful, after 7 failures (hence 8 steps are required for differentiation), at most 50 comparisons are necessary for differentiation, so the computation time becomes:

$$T' = 8TR' + 8TA + 400TC' \tag{4}$$

This comparison between T and T' shows that the difference between the two approaches is in the number of comparisons necessary to differentiate an object. There are fewer in the previous approach because the criteria of shapes stored in the database are used for comparisons, whereas only the shapes themselves are used in the proposed approach.

D. Memory footprint

In the proposed approach the memory of a cell is occupied with the programs to generate arrays for communications, replace sub-trees with the received trees and to transform the array to change the root to the northernmost and westernmost point so that another program can calculate the criteria. It also needs to store model data. The memory for these programs cannot be easily determined, since it depends on the implementation and the system, but it is a constant amount, that is independent of the models.

We consider the database of models which has the biggest memory footprint. This database has to be uploaded in each cell.

In the previous approach, one shape needs 29 bytes ($\simeq 15 \times 15\ /\ 8$). The number of shapes for the circle, rectangle and "H" are 48, 248 and 428, as presented above. The memory for the matrices is 29×724 bytes. The memory for the criteria is 4×58 bytes because one criterion needs 4 bytes and the numbers of criteria are 14, 23 and 21. In total, the memory needed by the models is 21228 bytes. If criteria with a larger memory footprint are used, the results could be even higher.

The data for one model consists of a number of shapes. The memory needed by a shape S_i is the number of nodes of its tree:

$$MS_i = \begin{cases} 4 & \text{if } n = 1, \\ 3n + 2 & \text{if } n \geq 2. \end{cases} \tag{5}$$

where n is the number of cells covered by the object.

Let N_{Model} be the number of models and N_{Shape} be the number of shapes for the model. The memory needed by a model M_k is:

$$MM_k = \sum_{S_i \in M_k} MS_i, \quad i = 1, 2, \ldots, N_{Shape} \tag{6}$$

Therefore, the total memory for models is:

$$M = \sum_{i=1}^{N_{Model}} MM_i \tag{7}$$

As an example, suppose a model "H" covers less than 25 cells and has about 430 shapes. A shape needs 77 bits, cf. equation 5. Thus, it needs 10 bytes, while the model with all its shapes needs 4.3 Kbytes. Similarly, the circle and the rectangle need 0.5 Kbytes and 2.5 Kbytes respectively. Hence, the memory needed to store all the models' data is about 7.3 Kbytes. This is much less than in the previous approach.

It is possible to further reduce the memory needed. This can be done by storing fewer shapes. However, the reconstruction phase and the differentiation phase need to be repeated until the shape matching the object is found. Moreover, having fewer shapes stored reduces the probability of matching. For this reason, the fewer the stored shapes, the longer the computation time. For instance, 12 shapes with 30° rotations stored in the database, create a footprint of 1.6 Kbytes.

V. Conclusions and future works

This paper aims to improve the distributed control of the Smart Surface by adopting tree-structured arrays. Representing the shapes as tree-structured data reduces their memory footprint. This footprint is, however, too large to implement if all the shapes of the models are in the database. The number of shapes can be reduced, but the approach trades off the reduction of the memory footprint against the probability of the successful differentiation. Future work will therefore focus on the optimization of the number of shapes to be stored in the database, and also on the distribution of the database on different cells to allow a much smaller memory footprint. The tree-structured array would be a good basis of support for distribution of shapes in the database.

VI. Acknowledgments

We would like to thank all the Smart Surface and Smart Blocks groups and in particular Guillaume Laurent et Nadine Piat. This work was supported in part by ANR (ANR-06-ROBO-0009 and ANR-2011-BS03-005-01).

References

[1] J. E. Luntz, W. Messner, and H. Choset. Distributed manipulation using discrete actuator arrays. *The Int. Journal of Robotics Research*, 20(7):553–583, 2001.

[2] M. Ataka, B. Legrand, L. Buchaillot, D. Collard, and H. Fujita. Design, fabrication and operation of two dimensional conveyance system with ciliary actuator arrays. *IEEE Trans. on Mechatronics*, 14:119–125, 2009.

[3] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. Single-crystal silicon actuator arrays for micro manipulation tasks. In *IEEE Int. Workshop on Micro Electromechanical Systems*, pages 7–12, 1996.

[4] J. W. Suh, R. Bruce Darling, K.-F. Böhringer, B. R. Donald, H. Baltes, and G. T. A. Kovacs. CMOS integrated ciliary actuator array as a general-purpose micromanipulation tool for small objects. *Journal of Microelectromechanical Systems*, 8(4):483–496, 1999.

[5] G. Bourbon and P. Minotti. Toward smart surfaces using high-density arrays of silicon-based mechanical oscillators. *Journal of Intelligent Material Systems and Structures*, 10:534–540, 1999.

[6] C. Liu, T. Tsao, P. Will, Y.C. Tai, and W.H. Liu. A micromachined permalloy magnetic actuator array for micro robotics assembly systems. In *Int. Conf. on Solid-State Sensors and Actuators*, 1995.

[7] G. J. Laurent, A. Delettre, and N. Le Fort-Piat. A new aerodynamic traction principle for handling products on an air cushion. *IEEE Trans. on robotics*, 27(2):379–384, 2011.

[8] P.-J. Ku, K. T. Winther, and H. E. Stephanou. Distributed control system for an active surface device. In *IEEE Int. Conf. on Intelligent Robots and Systems*, pages 3417–3422, 2001.

[9] J. Luntz and H. Moon. Distributed manipulation with passive air flow. In *IEEE Int. Conf. on Intelligent Robots and Systems*, pages 195–201, 2001.

[10] K. Varsos and J. Luntz. Superposition methods for distributed manipulation using quadratic potential force fields. *IEEE Trans. on robotics*, 22(6):1202–1215, 2006.

[11] A. Berlin, D. Biegelsen, P. Cheung, M. Fromherz, D. Goldberg, W. Jackson, B. Preas, J. Reich, and L.-E. Swartz. Motion control of planar objects using large-area arrays of MEMS-like distributed manipulators. *Micromechatronics*, 2000.

[12] S. Konishi and H. Fujita. A conveyance system using air flow based on the concept of distributed micro motion systems. *IEEE Journal of Microelectromechanical Systems*, 3(2):54–58, 1994.

[13] Y. Fukuta, Y.-A. Chapuis, Y. Mita, and H. Fujita. Design, fabrication and control of MEMS-based actuator arrays for air-flow distributed micromanipulation. *IEEE Journal of Microelectromechanical Systems*, 15(4):912–926, 2006.

[14] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2-4):210–225, 1993.

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

[16] L. Matignon, G. J. Laurent, N. Le Fort-Piat, and Y.-A. Chapuis. Designing decentralized controllers for distributed-air-jet, MEMS-based micromanipulators by reinforcement learning. *Journal of Intelligent and Robotic Systems*, 59(2):145–166, 2010.

[17] K. Boutoustous, G. J. Laurent, E. Dedu, L. Matignon, J. Bourgeois, and N. Le Fort-Piat. Distributed control architecture for smart surfaces. In Ren C. Luo and Hajime Asama, editors, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23, pages 2018–2024, Taipei, Taiwan, October 2010. IEEE.

[18] R. Huang, J. Ma, and Q. Jin. A tree-structured intelligence entity pool and its sharing among ubiquitous objects. In *Proc. Int. Conf. Computational Science and Engineering CSE '09*, volume 2, pages 318–325, 2009.

[19] K. Boutoustous, E. Dedu, and J. Bourgeois. An exhaustive comparison framework for distributed shape differentiation in a MEMS sensor actuator array. In *International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 429–433, Krakow, Poland, July 2008. IEEE computer society press.

[20] D. El Baz, V. Boyer, J. Bourgeois, E. Dedu, and K. Boutoustous. Distributed part differentiation in a smart surface. *Mechatronics*, pages 1–9, To appear.

[21] C. A. da Costa, A. C. Yamin, and C. F. R. Geyer. Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, 7(1):64–73, 2008.

[22] J. Ma. Smart u-things–challenging real world complexity. In *IPSJ Symposium Series*, 19, pages 146–150, 2005.

[23] J. Ma, L.T. Yang, B.O. Apduhan, R. Huang, L. Barolli, and M. Takizawa. Towards a smart world and ubiquitous intelligence: a walkthrough from smart things to smart hyperspaces and ubickids. *International Journal of Pervasive Computing and Communications*, 1(1):53–68, 2005.

[24] M.-W. Feng, S.-L. Wen, K.-C. Tsai, Y.-C. Liu, and H.-R. Lai. Wireless sensor network and sensor fusion technology for ubiquitous smart living space applications (invited paper). In *Proc. Second Int. Symp. Universal Communication ISUC '08*, pages 295–302, 2008.

[25] Y.-S. Jeong, E.-H. Song, G.-B. Chae, M. Hong, and D.-S. Park. Large-scale middleware for ubiquitous sensor networks. *IEEE Intelligent Systems*, 25(2):48–59, 2010.

[26] M. Weiser. The computer for the 21st century. *IEEE Pervasive Computing*, 99(1):19–25, 2002.

[27] J. Ma, Q. Zhao, V. Chaudhary, J. Cheng, L. Yang, R. Huang, and Q. Jin. Ubisafe computing: Vision and challenges (i). *Autonomic and Trusted Computing*, pages 386–397, 2006.

[28] K. Boutoustous, E. Dedu, and J. Bourgeois. A framework to calibrate a MEMS sensor network. In *Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing*, volume 5585 of *LNCS*, pages 136–149, 2009.