Semi-Customizable Gestural Commands Approach and its Evaluation

LI PeiYu[a] Renau-Ferrer Ney[b] Anquetil Eric[a] Jamet Eric[c]
[a]: IRISA/INSA Rennes [b,c]: University Rennes 2
Rennes, France
[a]: {pei-yu.li, eric.anquetil}@irisa.fr [b]: ney.renau-ferrer@irisa.fr [c]: eric.jamet@uhb.fr

***Abstract*—Pen-based interfaces allow users to interact with the help of a stylus and/or fingers. Thanks to a recognition system, users can execute commands by drawing gestures. Because of the complexity of the software, the set of gesture commands can be very large. Consequently, it becomes difficult for the users to remember all the commands. In this paper we introduce "Semi-customizable gestural commands". The idea is to take into account the commands usage frequency: to facilitate the memorization of the most used commands we leave users the freedom/liberty to define their associated gestures (by) themselves. The remaining gesture commands will be automatically generated on the base of the defined gestures according to their hierarchical categorization. Several comparative tests demonstrate that this new approach improved the progressive memorization of the gestural commands.**

Gesture commands; usage; incremental learning (key words)

I. INTRODUCTION

Pen-based interfaces allow users to draw strokes to run a command. These gesture commands offer an efficient and intuitive Human-Machine communication which is more and more researched nowadays [1][2][3][4][5][6][9][10]. For instance, we can associate a "C"-shaped gesture to Copy, a "V"-shaped gesture to Paste. In real software, the definition and the memorization of gestures associated to every command becomes a real challenge. Today the graphic gesture recognizers are stronger and more competitive; some of them can adapt and learn incrementally the user style [12][13]. The problem comes from the side of the user, who needs to learn all the gesture commands. Even if we try to associate intuitive and significant gestures to commands when it is possible, or to leave user the freedom to define the gestures by himself, this work becomes tedious when the number of gesture commands becomes important. To solve this problem, Kunterbach proposed the "Marking Menus" [4]. The main idea is to help user's memorization by manipulating menus which lead to final gestures corresponding to commands. The Marking Menus allow two kind of utilization: novice mode where the user will be guided by menus to run a command; and expert mode where the user has already learned the gesture and no menu will be displayed, so his gesture will be sent to a recognizer to be identified.

Since Marking Menus have met a big success, a lot of variants have been proposed. Above all, Hierarchical Marking Menus [5] allow more levels in menus so that the gesture set increases, but at the same time, it is more difficult for the recognizer. To keep a reasonable accuracy, Zhao et al. [9][10]proposed Zone and Polygon Menus where the user should draw several straight strokes to execute one command. Consequently, the set of induced gestures is constrained and cannot include the natural drawing habits, i.e. cursive gestures. Another interesting approach is Octopocus [2]. It introduces fluid and cursive gestures with a one-level menu. The induced gestures are diverse, contrary to other approaches who allow only polygonal lines which are neither natural, nor fluid. In novice mode, it guides users throughout the drawing of strokes with a colorful dynamic menu. This menu gives continuous feedback to the user. However, Octopocus allows only one level for the menu and the set of gestures is predefined and not customizable by the user. Moreover, there is no logic between each gesture and its associated command.

In our previous work, we first proposed the Continuous Marking Menus (CMM) [3] to improve the expressiveness of marking menus and to admit a vocabulary of cursive, fluent handwritten gestures. After some tests to evaluate this concept [6], we did prove that CMM helps the learning of gestures, when the original "help" displayed on the side of the software was not so helpful. Nevertheless, after the experiments, some users have been frustrated because they could not personalize some gesture commands. Based on these critics, we are convinced that it is important to give users a freedom degree to customize their gesture and it is also important to reduce the constraints on drawing the gesture as much as possible. Consequently, we focus on the new objective to answer these principal properties:

- *leave the gesture form as natural as possible;*
- *allow user to choose gestures associated to frequent commands;*
- *keep a semantic meaning and a hierarchy which can help the memorization;*
- *Use some advantages of Marking Menus to ease the learning.*

The results of this analysis are presented in this paper: we design the concept of "Semi-Customizable Gestural Commands (SCGC)" approach. The SCGC's originality is to consider the usage frequency of commands to ease the learning of commands gestures. At the same time, we leave user the freedom to define his gestures for the predominant

978-0-7695-4774-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICFHR.2012.267

commands, and the remaining gestures are automatically generated based on the gesture of the predominant command. The objective is to get natural and cursive gestures which are easy for the user to learn and memorize at the same time.

The rest of the paper is organized as follows. We introduce at first the SCGC approach. Then we expose the incremental gesture recognizer [12][13] we have used to conduct experiments. Finally we present a complete experimentation which involves more than twenty users in three comparative tests to evaluate the impact of SCGC approach in terms of learning and memorization of the gestures by the users. We also report, in parallel with what the user learnt, the result of the incremental learning of the gesture recognizer used in these experiments.

II. SEMI-CUSTOMIZABLE GESTURE COMMANDS

According to the users' feedback in our previous works, we estimate that it is really important to keep users defining some gestures themselves to help their memorization (particularly for the most frequent commands). Moreover, in real software, all commands do not have the same use frequency. For instance, the "copy" command is used more often than the "print" command. In general, user wants to memorize the most used commands at first. The more we use a command, the faster we learn it. This is why we introduce a new kind of commands: the Semi-Customizable Gesture Commands (SCGC). The SCGC approach leaves the user the freedom to design gestures for their most used commands, and then generates all the derived gesture commands automatically. It avoids the user the tedious task of defining all the gestures. To make the generated gestures meaningful, we group the commands by family, like Marking Menus. More precisely, we ask the user to define the gesture for one predominant command per family, and the gestures for the other members will be automatically generated.

To implement automatic generation of gesture commands, all the gestural commands are modeled by spline to formalize the prototypes of the gestures. All derived gestures are then generated by a continuous deformation of the predominant prototype. The next sections present the different steps to formalize the drawing of the gestural commands.

A. Polygonal approximation

The first step consists in a polygonal approximation of the drawing of the gestural commands. We use the polygonal approximation algorithm defined in [7]. Given a collection of N points and a constant M, this algorithm can find the M points which form a polygon that follows the best curve defined by the N points collection. The quality of the polygonal approximation is quantified by an error indicator. We can find the optimal polygonal approximation according to this indicator. For instance, in Figure 1, the black curve is the original stroke drawn by user. The step 1 corresponds to polygonal approximation. These points are polygons corners

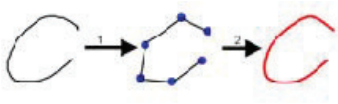


Figure 1. Prototype creation.

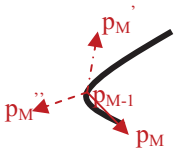


Figure 2. Example of generation of 2 new last control points.

which are automatically extracted with the polygonal approximation algorithm. Here we have $M=6$.

B. Canonical spline

The M points extracted by the polygonal approximation are used as control points to define a canonical spline to represent the original drawing.

A canonical spline is a cubic polynomial who aims to imitate the original curve by defining several control points to regularize the drawing [8]. Unlike the Bézier curve, the canonical spline passes through all its control points. The implemented canonical spline algorithm is based on [8] . In Figure 1, the second step corresponds to a canonical spline definition.

C. Automatic gesture generation

The generation for gesture commands is based on a continuous deformation of the gesture's end.

Let's note the M control points of the gesture that the user defined: $p_1, \ldots, p_M$. For the other members of the same family, we keep the same beginning, i.e.: $p_1, \ldots, p_{M-1}$. We move simply the last control point to generate new gestures. Thanks to the properties of splines, the newly generated curves will stay fluid and cursive.

Figure 2 illustrates the principles of the automatic generation of all the alternative commands of a same family. The black stroke presents the canonical curve of the original curve drawn by user. For a family who has three commands, we take the coordinates of the last two points: p_{M-1} and p_M. We generate the new coordinates of p_M' (new last control point for one new command) and p_M'' (new last control point for the other command), respectively:

$|p_{M-1}\,p_M| = |p_{M-1}\,p_M'| = |p_{M-1}\,p_M''|$ and

$\angle p_M\,p_{M-1}p_M' = \angle p_M p_{M-1}p_M'' = \angle p_M''\,p_{M-1}p_M'$.

The control point collection for the two new commands of the family are $(p_1, p_2, \ldots, p_{M-1}, p_M')$ and $(p_1, p_2, \ldots, p_{M-1}, p_M'')$ respectively.

Figure 3 illustrates a real use case. In step 1, after getting the original curve of the user, a canonical spline is generated with $M=6$. In step 2, the other two gesture commands of the same family are generated automatically.

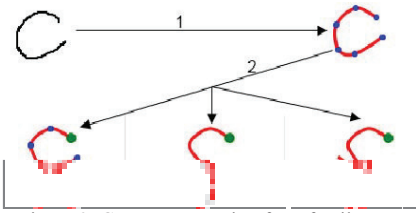


Figure 3. Gesture generation for a family.

III. RECOGNIZER

In SCGC approach, for gesture recognition, we adopt a pattern recognition approach adapted to this application. The recognizer needs to be able to learn with very few training samples because the set of gestures will be different for each user. Input gestures are described by a reduced set of 15 features, and classification is executed by an incremental lightweight Fuzzy Inference System that requires few training samples [12][13].

A. Feature extraction

Input gestures are first linearly rescaled so their maximal dimension is 64, in order to make the recognition of gesture scale-independent. The pen trajectory is then resampled so that the spatial distance between points is always equal to 8, to avoid sensitivity with respect to speed variations in the writing. We finally extract a set of 15 features, combining dynamic (time-based) and visual (image-based) features, so as to handle different properties of the patterns. Some retained features are inspired from Rubine's features [11]: trajectory length, initial angle, first point to last point vector. The position of the first point and the last point of the trajectory with respect to the gesture's bounding box is also described. As visual descriptors, we extract zoning features from a regular 3x3 grid partition of the bounding box.

This diversity of features is essential for some tests (see section 4) where the recognition of "unpredictable" user-defined gestures is needed. While no optimization is possible, given this "open-world" problem, it is hoped that the chosen feature set can properly distinguish between a sufficiently large set of classes defined by the same user.

B. Fuzzy inference system

Most of classical feature-based recognition methods (neural networks, statistical recognizers, Support Vector Machines…) involve a training phase that is executed offline. In our situation, however, the training phase has to be run online, since the gestures chosen by the user have to be learned by the classifier. Another constraint is the few available training samples, since the user should not need to perform more than 3 samples per gesture. For these reasons, we chose to use an Incremental Fuzzy Inference System, a competitive discriminative classifier that requires very few training samples and computing resource. This system is designed to be integrated into applications which need customization aspect. Actually, the system can be used as an evolving recognizer, with lifelong learning and handling of new classes of gestures (see more details in [12][13]). The reduced size of the feature set (only 15 features) is also essential to obtain better classification performance with only few training samples.

IV. EXPERIMENTS

To evaluate our SCGC approach, we made several experiments. On first step, we made some pre-tests to see users' reaction for this new approach on a tablet. After having observed some difficulties on manipulation, we adapted our approach to make the learning even much easier as described in section 4.1. On second step, we fixed the objectives of these experiments: we wanted to evaluate SCGC in a real piece of software's context (ecological frame). It means that we would like to see user's attitude without telling them that it is a test. We wanted to see how gestures are learnt unconsciously by users. We did not want them to feel obliged to memorize the gestures which is different from a lot of tests made before [1][2][9][10]. So we made three different comparative tests: test1 where the user chose all his gestures; test2 (SCGC approach) where the user chose the root gesture and the relative gestures were automatically generated; and test3 (SCGC without personalization) where root gestures are pre-defined.

A. SCGC adaptation

We made some pre-tests to collect users' observations on SCGC. We realized that it is difficult for a user to remember the relative direction of a spline's end in one family. To overcome this difficulty, we fix the spline's end in three absolute directions: down, right and left. So instead of defining gesture for the predominant command in a family, the user defines a root gesture for the family. Figure 4 illustrates this change (more examples in Figure 6).

B. Presentation of the application

The application context of the tests we performed took the shape of a pen-based picture editor program with which users can perform various actions using graphic gestures. The aim is to obtain an ecological frame, allowing the user to be confronted with a real case of gestural commands' use. Moreover, whereas the test protocols involved the user only being asked to draw a series of commands into a box, a real use of a concrete application does not allow placing the user in an "artificial" learning situation. The ecological frame we used allows deceiving the user by making him think that the purpose of the test is not to memorize the commands but to test the use of graphic commands applied to picture manipulation. The goal is to obtain user learning rates in a passive learning context.

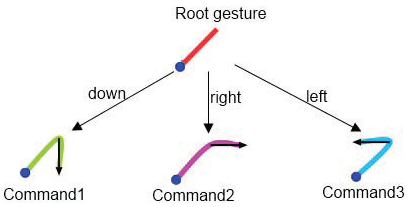


Figure 4. New SCGC for a gesture family.

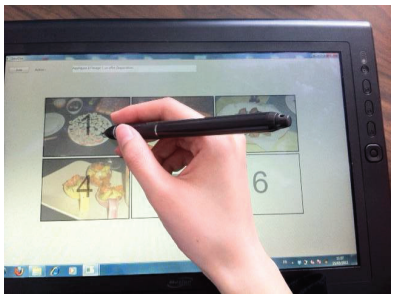
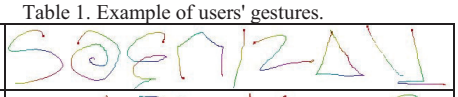
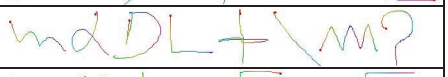
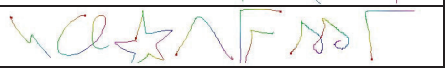


Figure 5. Test's picture.

Table 1. Example of users' gestures.

Display icons	
Display normal	
Personal folder	

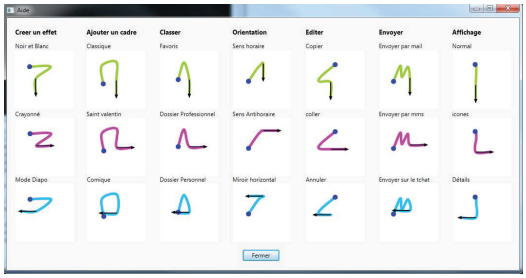


Figure 6. "help" window with the gesture set.

C. *Experimental protocol*

Users are distributed in 3 groups; each group corresponds to a different kind of experimentation. Users from group 1 were asked to choose a gesture for each of the 21 commands. For group 2, we used the SCGC approach where users only have to choose a root gesture for the 7 families of commands, and then the system automatically generates the 21 corresponding gestures. Finally for group 3, the 7 root gestures and the corresponding 21 gestures were forced onthe users. For each group, users were asked to draw each gesture 3 times in an initialization phase. We present a variety of gestures from users from group 1 for 3 different commands in Table 1.

Experimentation itself is divided into 2 use phases and 2 test phases. Use phases are made of 34 questions, each of them corresponding to a command. Moreover, some commands were asked more often than others. 6 of them were asked 3 times by use phase, 4 were asked twice and 8 were asked once. The remaining 3 commands were asked during use phases. During the use phases, user had access to a help window with a table of all the gestures (Figure 6).

During the test phase user had to achieve once each of the 21 gestures in a random order. During test phase, user could not display the help windows. To summarize, the protocol follows the pattern:

a. *Initialization: user draws each gesture 3 times*
b. *Use phase1: user answers the 34 questions*
c. *Test phase1: user draws the 21 gestures*
d. *Use phase2: user answers the 34 questions*
e. *Test phase2: user draws the 21 gestures*

When a gesture is made, it is sent to the recognizer. If the gesture is correctly recognized, the visual effect of the command applies so the user can be informed that he has successfully performed his gesture, and then another question is asked. On the other hand, if the answer of the recognizer is different from the gesture that was asked, we must know if the recognizer failed or if the user performed a wrong gesture. So, in this situation, a window will pop up to show the user the asked gesture and the gesture he drew. In this window, the user has to state whether or not he has performed the right gesture. If he drew the good gesture, the visual effect applies and the test goes one. However if he drew a wrong gesture, he will then have to try again. When a gesture is correctly recognized, the classifier uses it to strengthen its learning for this class. When the classifier is mistaken there are two possibilities: if the user states that he has done the good gesture, then it's used to strengthen the gesture that was asked, instead of the gesture which has been recognized. If the user states that he did a wrong gesture, and then nothing happens, the user has to draw it again.

D. *Results*

We present here the results of 23 users divided into the 3 groups: 9 in group 1, 7 in group 2, 7 in group 3. The group 1 dataset contains 1017 gestures divided into 189 classes (21 per user) with 7 to 11 examples per class. The group 2 dataset contains 861 gestures divided into 147 classes (21 per user) with 7 to 11 examples per class. The group 3 dataset contains 861 gestures divided into 21 classes with 49 to 77 examples per class (7 to 11 per user).

1) User learning curve: Figure 7 shows the average score for each group. This score represents the percentage of gestures correctly realized by the users during each testing phase. The dotted curve corresponds to the group 1, the dashed one to the group 2 and the solid curve shows the score for the group 3.

We can see that the average score of the group 1 for test phase 1 is the higher (83%). This is most likely due to the fact that users can here choose for each command a familiar gesture. Also, users in this group have the possibility to choose gestures that have a very strong semantic link with the commands (examples include drawing a square for the command "classic frame", a heart for "valentine frame" or an @ for "send by email"). Consequently, many gestures are

already “known” by the user since the very beginning of the test. On the other hand, not all commands can be associated to a “semantic” gesture. Moreover, memorizing 21 different gestures remains a hard task. As we can see, final scores of group 1 will finally be weaker than the scores of group 2. We can notice that group 1 presents the worst learning progression rate with 0.6%, while in group 2, the users progressed at a 19% rate and in group 3, 33%.

Now let us take a look at the results concerning group 2 and 3. Those two groups use the same system of gestures generation from a root family gesture. Group 2 corresponds to the SCGC approach where the users can choose their root gestures while in group 3, they are imposed. As we can see, even if the scores of group 2 are lower at the beginning, they finally exceed those of group 1 with a score of 93%. This shows that even if users cannot customize their gestures as much as they would like in order to make the task easier, only having to memorize 7 gestures and the 3 variations is a much easier task than memorizing 21 gestures, even if customized gestures seem to be better known at first. Group

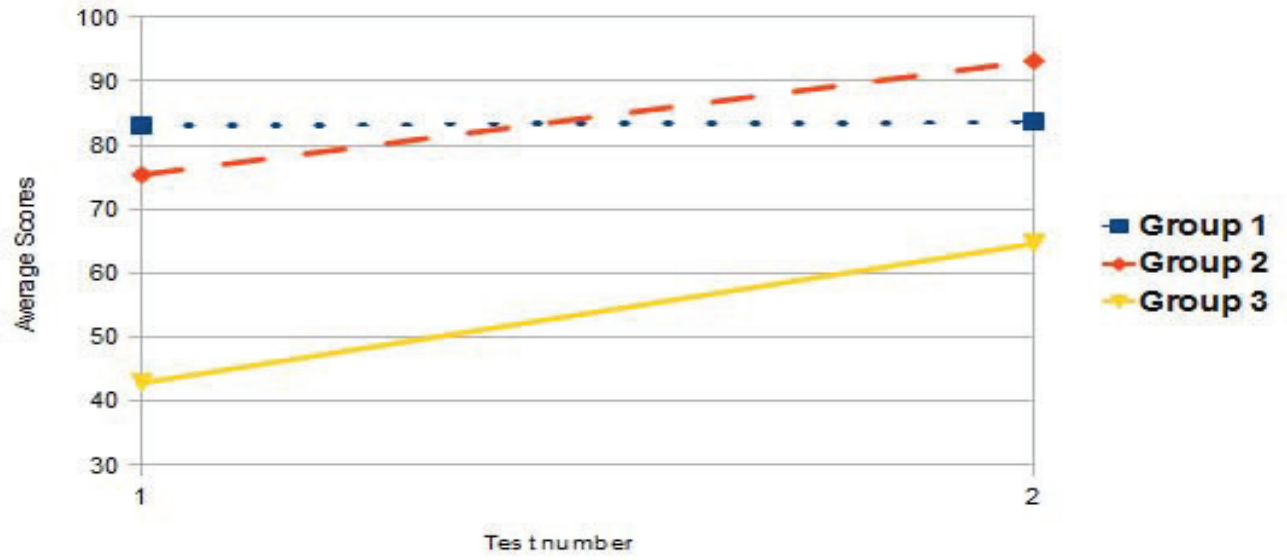


Figure 8. Percentages of gestures correctly known by the user for the 2 test phases.

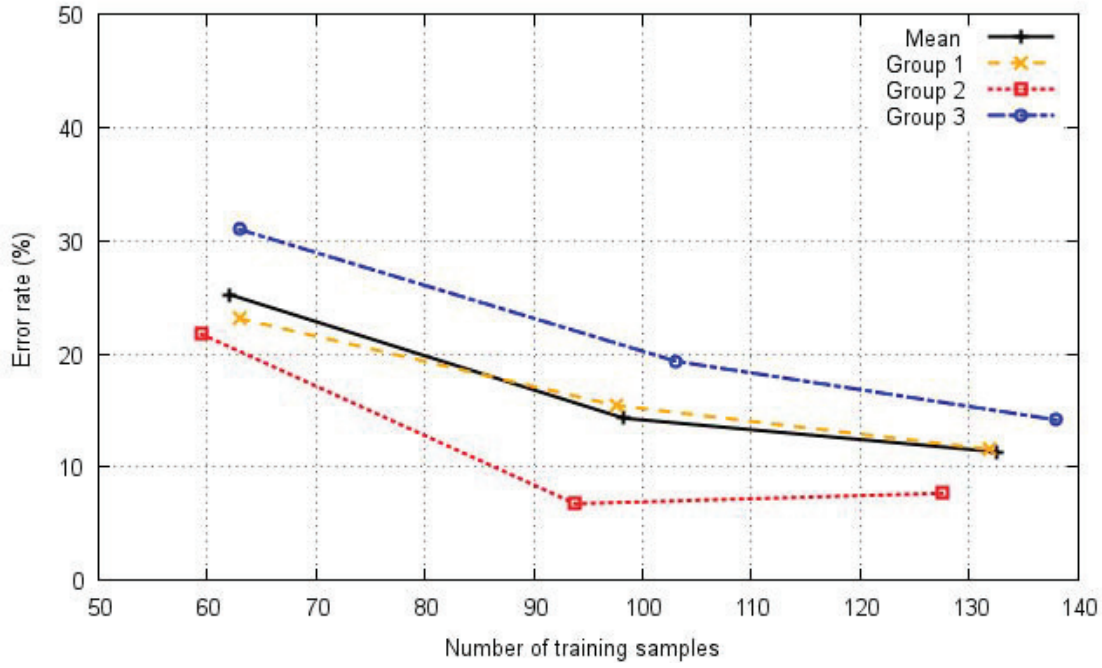


Figure 7. Classifier's learning curve.

3 is a perfect illustration of this principle. Of course this group presents the lowest scores from the beginning to the end. This is easily explainable by the fact that all gestures and root gestures are unknown for the user. However the use of the family organized gestures with our method of gestures generation allow this test to present a high progression rate of 33.7%.

2) Classifier's learning curve: At the same time, to learn how the user behaved in front of these tests, we also wanted to see the incremental classifier's behavior. We then took the gestures obtained during the two test phases as a test database (without the three commands which were not asked to draw during the use phases). This test database is used to evaluate the classifier's performance at the end of the initialization phase, at the end of use phase 1 and at the end of use phase 2. So we obtained three control points to see the incremental learning of the classifier. We obtained the learning curve of the classifier reported in Figure 8. The control point 1 begins at a different place for these curves because users can choose not to repeat 3 times all gestures during the initialization phase. The difference at point 2 or point 3 is due to questions asked more than once when the user did not draw the good gesture. On the mean curve, we can find that after the initialization phase, the recognizer has a 25.2% error rate on the test database. It has a 13.4% error rate after use phase 1. Finally it has a 11.3% error rate after use phase 2. We conclude that even with few gesture samples (7 to 11 for each command), the incremental recognizer has learned progressively the gestures from the users, which explains the decrease on the error rate.

V. CONCLUSION

In this paper we present the new SCGC approach to make fluid and natural gesture commands easy to memorize and to learn for the user. The reported experiments demonstrate that a user has better memorization with the SCGC approach than by defining all the gestures himself. We conclude that the learning progression of the user is faster with the SCGC approach. We analyzed the users' learning curve and the classifier's learning curve at the same time: we show that the fuzzy incremental recognizer is really accurate on this kind of software scalable. All the material of the reported experiments (database) will be soon available on our research team's website (http://www.irisa.fr/intuidoc/).

ACKNOWLEDGMENT

During this experiment, we received a lot of help from the platform LOUSTIC. We would like to express our thanks to Caroff Olympe and Uhde Alarith who organized the test séances. We would like also to give our thanks to Manuel Bouillon who helped us a lot on classifier's learning curve.

REFERENCES

[1] Bailly, B., Lecolinet, E., Nigay, L. "Wave Menus: Improving the Novice Mode of Hierarchical Marking Menus," Proc. INTERACT'07. Springer: 475—488, 2007.

[2] Bau, O. and Mackay, W.E. "Octopocus: A Dynamique Guide for Learning Gesture-based Command Sets," Proc. UIST 2008, ACM Press: 37-46, 2008.

[3] Delaye, A., Sekkal, R., and Anquetil, E. "Continuous Marking Menus for learning Cursive Pen-based Gestures," Proc. IUI 2011, ACM Press: 319-322, 2011.

[4] Kurtenbach, G. "The design and evaluation of marking menus." PHDThesis, 1993.

[5] Kurtenbach, G. and Buxton, W. "The limits of expert performance using hierarchic marking menus," Proc. INTERCHI (1993), IOS Press: 482-487, 1993.

[6] Li, P., Delaye, A. and Anquetil, E. "Evaluation of Continuous Marking Menus for Learning Cursive Pen-based Commands," Proc. IGS 2011: 217-220, 2011.

[7] Perez, J. and Vidal, E. "Optimum polygonal approximation of digitized curves," Pattern Recognition Letters 15: 743-750, 1994.

[8] Petzold, C. "Canonical Splines in WPF and Silverlight." http://www.charlespetzold.com/blog/2009/01/Canonical-Splines-in-WPF-and-Silverlight.html.

[9] Zhao, S., Agrawala, M. and Hinckley, K. "Zone and Polygon Menus: Using Relative Position to Increase the Breadth of Multi-stroke Marking Menus," Proc. CHI 2006, ACM Press: 1077-1086, 2006.

[10] Zhao, S. and Balakrishnan, R. "Simple vs. Compound Mark Hierarchical Marking Menus," Proc. UIST 2004, ACM Press: 33-42, 2004.

[11] Rubine, D. "Specifying gestures by example."

[12] Abdullah Almaksour, Eric Anquetil. "Improving premise structure in evolving Takagi-Sugeno neuro-fuzzy classifiers," Evolving Systems, 10.1007/s12530-011-9027-0, 2:25-33, 2011.

[13] Abdullah Almaksour, Eric Anquetil, Solen Quiniou, Mohammed Cheriet. "Personalizable Pen-Based Interface Using Lifelong Learning," Proc. International Conference on Frontiers in Handwriting Recognition (ICFHR'12), 188-193, 2010.