UsiGesture: an Environment for Integrating Pen-based Interaction in User Interface Development

François Beuvens and Jean Vanderdonckt
Université catholique de Louvain, Louvain School of Management
Louvain Interaction Laboratory, Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{francois.beuvens, jean.vanderdonckt}@uclouvain.be – Phone: +32 10 478525

Abstract— **Several algorithms have been developed for pen-based gesture recognition. Yet, their integration in streamline engineering of interactive systems is bound to several shortcomings: they are hard to compare to each other, determining which one is the most suitable in which situation is a research problem, their performance largely vary depending on contextual parameters that are hard to predict, their fine-tuning in a real interactive application is a challenge. In order to address these shortcomings, we developed UsiGesture, an engineering method and a software support platform that accommodates multiple algorithms for pen-based gesture recognition in order to integrate them in a straightforward way into interactive computing systems. The method is aimed at providing designers and developers with support for the following steps: defining a dataset that is appropriate for an interactive system (e.g., made of commands, symbols, characters), determining the most suitable gesture recognition algorithm depending on contextual variables (e.g., user, platform, environment), fine-tuning the parameters of this algorithm by multi-criteria optimization (e.g., system speed and recognition rate vs. human distinguishability and perception), and incorporation of the fine-tuned algorithm in an integrated development environment for engineering interactive systems.**

Keywords-Gesture recognition, pen-based interaction, user interface modeling Introduction.

I. INTRODUCTION

Pen-based gesture recognition [1,2,3] typically consists in interpreting hand-made marks, called strokes [4,5], made with a pointing device (e.g., a mouse, a stylus, a light pen) on a flat constrained vertical or horizontal surface (e.g., a table, a wall or a graphic tablet). Pen-based gestures are applicable to a large area of tasks (e.g., music editing, drawing, sketching, spreadsheets, web navigation, equation editing) in many different domains of activity (e.g., office automation [6], ambient intelligence [7], multimodal systems [8]) and a growing set of devices, ranging from smartphones to tabletop interaction. Pen-based gestures can even be considered across several platforms: starting on a smartphone and finishing on a tabletop [7]. When the locus of input is different from the locus of output (e.g., with a graphic tablet), gestures are drawn outside the main display, thus posing a visual discontinuity problem. When locus of input and output match, a risk of occlusion occurs since the gesture is drawn on top of the main display. The surface used for pen-based gestures is however used as a way to constrain the gesture, thus helping its recognition.

Pen-based gestures have received considerable attention in both research and development, namely for addressing the scientific problem of modeling, analyzing, learning, interpreting, and recognizing gestures in a large spectrum of setups. The large inclusion of pen-based gestures in widely-available interactive applications has however not reached its full potential due to at least the following reasons: designers and developers do not know which recognition algorithm to select from such as large offer, how to tune the selected algorithm depending on their context of use, and how to incorporate the selected algorithm into streamlined User Interface (UI) development in an effective and efficient way. Incorporating pen-based gestures may also involve using Application Programming Interfaces (APIs), libraries, toolkits or algorithm code that could be considered hard to use [9]. Consequently, in this paper, we do not address the scientific problem for modeling, analyzing, interpreting, and recognizing gestures. Rather, we are interested in integrating the results provided by this body of research [10,11,12,13,14,15] into streamlined UI development with device independence, extensibility, and flexibility.

The remainder of this paper is structured in five sections: the first section reviews the state of the art, the second describes the implementation and the functioning of the gesture recognition platform, and the third section describes the method for incorporating pen-based interaction in user interface development life cycle through a running example. Finally, a conclusion delivers the main points of this research and presents some future avenues.

II. RELATED WORK

A. Motivations for pen-based gestures

Pen-based gestures are appreciated by end users for several reasons: they are straightforward to operate [16], they offer a better precision than finger touch [17], which make them particularly adequate for fine-grained tasks like drawing, sketching. Most people found gestures quick to learn and to reproduce once learned, depending on properties [6,18]:

1) Iconicity («memorable because the shape of the gesture corresponds with this operation» [19]). When humans are communicating, they are using gestures to increase the understanding of the listener and obviously, the gesture usually means what the speaker is saying. Iconicity principle is based on this. It means that gestures that are designed are close to the

interpretation of this reality. For example, Fig. 1(a) depicts the "Delete" action by a pair of scissors, which denotes the activity of cutting something.

Figure 1. Some pen-based gestures for "delete" (a), "copy" (b), "u" and "v" letters (c), move forward (d), move back (e).

2) Learnability. Users sometimes forget gestures because they are numerous or because they are complex or not iconic. 90% and more participants held that pen-based gestures with visual meaningful related to commands are easy to be remembered and learned [16]. Another option suggested in [17] to increase the rememberability of the users was to represent a gesture as the first character of the command name. For instance, Fig. 1(b) depicts the "Copy" action because «C» stands for Copy, which is an alternative provided that shortcuts are available. If users spend their time for checking which pen-based gesture is convenient for executing which command in the manual, they will get bored soon.

3) Recognizability. Naive designers often create pen-based gestures that the computer viewed as similar and thus were difficult to recognize [18]. There is a trade-off between improving the gesture recognizability for the gesture recognizer and optimizing the recognizability for the end user. For instance, Fig. 1(c) compares two alternate designs for writing the "u" and "v" letters. The right one improves the system recognition rate of the system, but deteriorates the end user recognition (since the gesture looks like a square root), while the left one is the opposite: natural for the end user, hard for the system to differentiate.

4) Compatibility and coherence. Gestures [17] are also better learned and used when they are compatible and coherent. Gestures are best perceived when they are introduced in a uniformed and stable way. For instance, the gesture of Fig. 1(d) depicts "moving forward or right" since the gesture indicated the direction, which is natural to understand for the user, while Fig. 1(e) means the opposite direction.

B. State of the Art

Many tools may be used for getting support during the conception of user interfaces including gesture recognition.

iGesture [20] is aimed at helping developers selecting a suitable algorithm for specific needs, and additionally supports them to add new gesture recognition functionality to their application. Moreover, it brings support to developers wanting to design new recognition algorithms.

InkKit [12] is a sketch toolkit designed to support diagramming across a wide range of domains. It consists of an excellent user interface and a customizable recognition engine. The user interface has two main views: sketch pages and portfolios. On a sketch page the user can draw and write much as they would on a piece of paper, yet supported by usual computer editing functionality. The portfolio is a place where a collection of sketches is displayed and links can be used to establish relationships between sketches.voi

The *Georgia Tech Gesture Toolkit* (GT2k) [19] provides a publicly available toolkit for developing gesture-based recognition systems. It provides capabilities for training models and allows for both real-time and off-line recognition.

Quill [21] is a toolkit created to help designers to create pen-based gestures based on the Rubine's algorithm [22]. To use the Quill toolkit, we have to draw into different gesture categories all the gestures to be recognized in the future. A gesture category is a set of same gestures. The idea of the gesture category is to gather information about a same gesture which cannot be drawn exactly twice the same. Each gesture should be drawn several times into each category (10 to 15 times). Quill informs the user of the possible similarity of the newly drawn gesture category and the existing gesture categories in order to optimize the recognizer. When the gesture has to get recognized, the recognizer is called and computes once again the specific features of the drawn gesture and compare them with the values of the features of the «learned gestures». The recognizer returns the list of all the gestures sorted by decreasing order of similarity. The first item in this order is then the more-look-like gesture based on the proximity of the features values.

Satin [23] has been created for making effective pen-based user interfaces easier. The two important facets are: the integration of pen input with interpreters and the libraries for manipulating ink strokes. In a Satin sheet, the user can draw the gestures that are interpreted with the libraries that can handle the ink strokes. Satin then get back the recognized gesture of Quill.

DataManager [15] is a toolkit with the ability of automatically evaluating recognition algorithms. It is shipped with six different recognizes and allows a developer either reusing the provided algorithms or add its own algorithms in order to compare the performances.

The *Gesture and Activity Recognition Toolkit (GART)* [19] is a user interface toolkit designed to enable the development of gesture-based applications. GART provides an abstraction to machine learning algorithms suitable for modeling and recognizing different types of gestures. The toolkit also provides support for the data collection and the training process.

C. Motivations for integration in mainstrean UI development

In this subsection, the pieces of work presented in the state of the art are reviewed by referring to some of their shortcomings.

Accessibility. Pen-based gesture recognition algorithms are usually made accessible as APIs, libraries, toolkits, platforms or procedural code. A recognition algorithm may become available either in one or many of these heterogeneous forms, thus making their integration in application development a permanent challenge. When a same algorithm is found in

different sources, the source choice is even more challenging. Some platforms, like iGesture, InkKit or DataManager offer several algorithms though, but they are more intended for recognition benchmarking.

Hard Algorithm Selection. Certain platforms help the designer to create a gesture set (e.g., Quill) or to benchmark recognition algorithms in a general way (e.g., DataManager). But, to our knowledge, none of them is able to drive the designer through the different steps of selecting a recognition algorithm that is appropriate for a particular gesture set, of fine-tuning its parameters in order to optimize its performance, and for integrating this fine-tuning in UI development.

Device independence. In principle, recognition algorithms should be independent of any pointing device, like any type of stylus. In practice, they do not have all the required drivers for this purpose. GART is a toolkit supporting the development of pen-based gestures by providing an abstraction to machine learning algorithms suitable for modeling and recognizing different types of gestures, thus achieving also some level of algorithm independence.

Lack of extensibility and flexibility. It is usually very hard to extend the current implement with another recognition algorithm or another variant of an existing algorithm. Some platforms, like iGesture [20], InkKit [12] or Satin [22], already hold several recognition algorithms with room for extension, but this task is more intended for highly-experienced developers. In addition, they do not help in selecting which algorithm or interpretation of an algorithm is most appropriate for a certain task.

Incorporation in UI development. Most of the time, the incorporation of pen-based gestures should be achieved in a programmatic way, by hand, with little or no support [3]. This is probably one of the most challenging shortcomings. There is no continuity between the design phase of pen-based gestures and the development phase of incorporating the algorithms in the final interactive application. Wilhelm *et al.* describe a trajectory-based approach that is applied to support device independent dynamic hand gesture recognition from pen devices. Feature extraction and recognition are performed on data recorded from the different devices and transformed them to a common basis (2D-space).

This paper is original in that it provides a method (and a software support) for incorporating gesture recognition algorithms into mainstream software development, with device independence, extensibility, and flexibility.

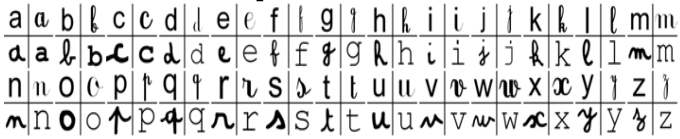
III. A Software Platform For Gesture Recognition

This section describes the gesture recognition platform through the different step the designer has to follow in order to integrate gesture manipulation in a user interface. The first step consists in choosing the gestures to be used in the interface. Other people must then reproduce these gestures in order to constitute a training set allowing the training of recognition algorithms. The third step is then to train and test the recognition algorithms. This step is supported graphically and allows the designer to train and test directly an algorithm for some parameters and one gesture. Additionally, a more rigorous algorithm comparison is proposed by relying on statistical analyses in order to highlight more features useful to determine the most suitable algorithm and settings for the specific purposes of the designer.

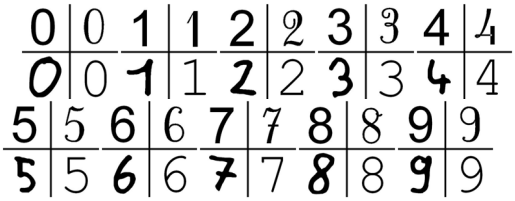
A. Choosing the gestures

The final goal of the platform is allowing the designer drawing his own gestures and helping him in this task. However, for the moment, this possibility is not available. A predefined gesture set is proposed, instead. In order to determine this set, we could have selected an existent database. But even if databases such as IAM [24] containing lots of samples can be found, we had to collect the new data ourselves. Indeed, a database like IAM is based on the contribution of more than 500 persons, more than 1500 scanned sheets, 10 000 isolated lines of text and 100 000 isolated words, but is only usable for offline recognition. Other databases such as UniPen [25], UCI [26], UJIPen [27] or Graffiti [28] would be more useful as they provide much more information such as the complete dots set describing the gesture, temporal information, pressure, pen inclination, etc. However these databases only include letters and numbers. Since we want to propose more diversity, we had no other choice than building our own database so as to integrate geometrical shapes and action commands, in addition to the letters and numbers. The choice of the gestures to obtain was dictated by the feasibility of the encoding. It was not possible to ask to the participants to spend more than one hour sketching gestures. The set of gestures was thus reduced to its minimum, while considering all the various gesture types. Here are the templates used by the training platform (four representations are depicted for each template in order to encourage participants to use their own writing and not copy the image they see):

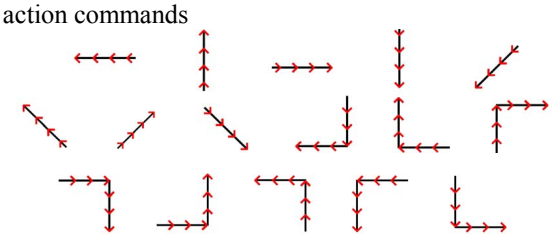
- The 26 letters of the alphabet



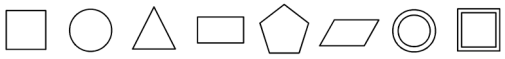
- Figures from 0 to 9



- 16 action commands



- 8 geometrical shapes



B. Building the training set

To ensure a sufficient training of the algorithms, the participants were asked to provide 10 samples for each gesture through the platform window presented in Fig. 2(a). A total of 30 people participated to the database development. Each gesture was then reproduced 300 times. The participants were asked to produce the gestures randomly, in order to avoid quality distortion for some gestures. Indeed, during the experiment, the user is likely to take confidence and goes faster and faster trying to end quicker. Mixing the data makes it possible to distribute the various moments of attention and lassitude, but also to avoid a biased "noise". Rules were elicited for gesture acquisition:

- Participants were requested to avoid making erasures. If necessary, they were asked to erase the complete gesture and to start again. The dimension of the gestures and their position on the screen were free.
- Participants were told it was not a contest. The main purpose was to be as natural as possible.
- Action commands were represented with a black line, and red arrows to indicate the drawing direction.
- Letters and figures had to be drawn with natural writing, with no constraints on the way to write them.
- Geometrical shapes could be drawn with several lines (except for the circle). Participants were informed that the precision and artistic aspects were secondary.

The physical setup was similar for all participants. The computer used was equipped with an Intel Pentium M 1.6GHz CPU, 2Gb DDR of RAM and a 15 inches LCD screen with a resolution of 1400x1050. The pen-based device was an A4Wacom Intuos3 Pen Tablet (9x12 inches) with a precision of 200 lines per millimeter. All details and screenshots related to this procedure are accessible at https://sites.google.com/site/comparativeevaluation/data-collection.

C. Training and testing the recognizers

Thanks to the training set, the designer may proceed to the training and testing of the recognizers. The algorithms selected and included in the platform are for the moment Rubine [22], One Dollar [9], Levenshtein [29] and stochastic Levenshtein [30] but it is possible to add new algorithms. Fig. 2(b) shows the platform window allowing performing a single test in order to get a quick feedback on the behavior of the algorithm. Indeed, the designer can sketch any shape in the drawing area, select an algorithm and settings, choose the training gesture set, specify general recognition settings, train the algorithm, launch the test, and check in the information area the result of the recognition. The available options are described in the rest of the subsection.

For each of the algorithms already present in the platform, the designer has the possibility to edit all the parameters to be

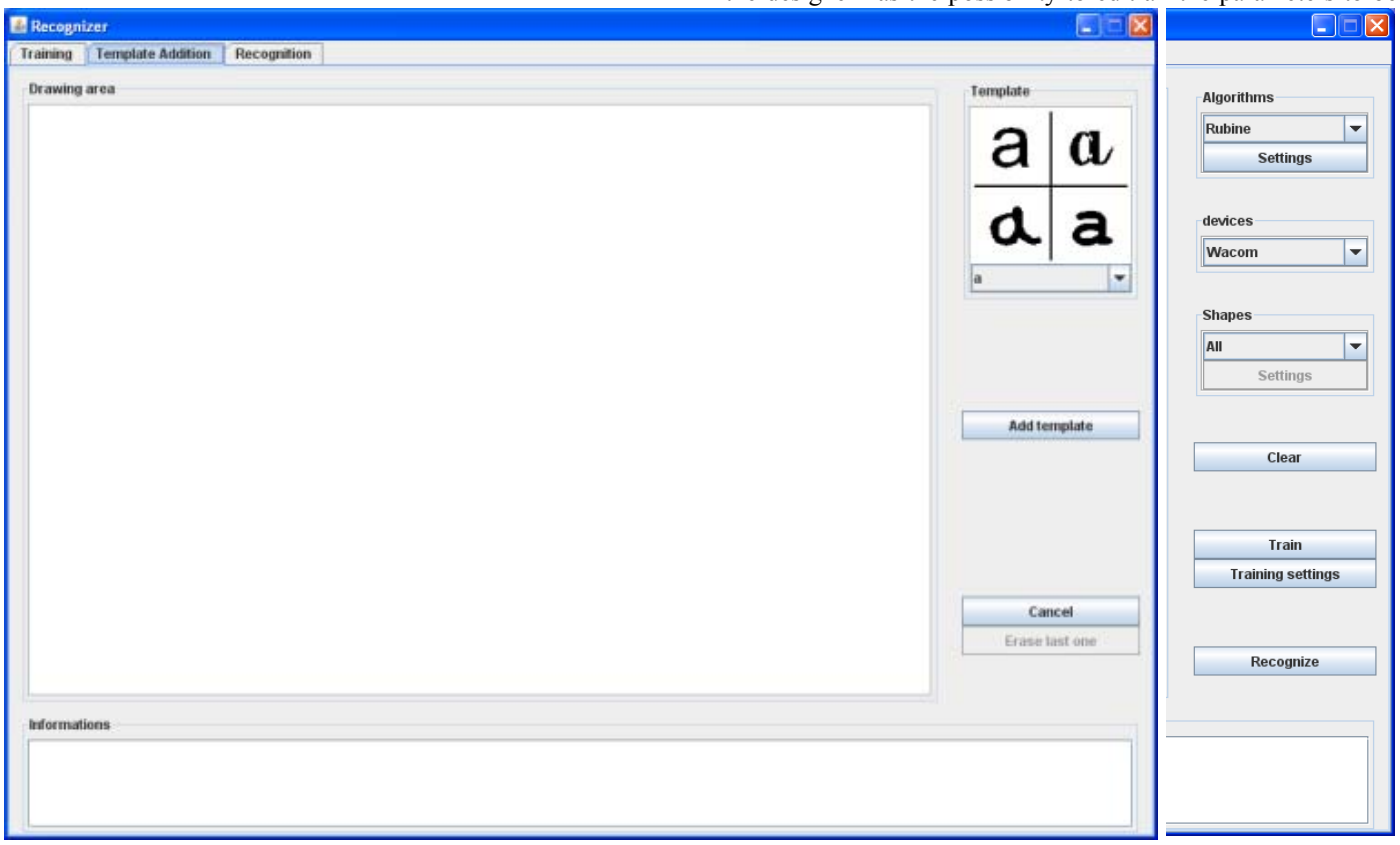


Figure 2. Training set building (a), Training and testing (b).

used for the recognition, including pre-handlings that are generally not used with a specific algorithm. Our tool allows thus evaluating the standard approach, but also the impact of different pre-processing applied to algorithms that are usually not provided. Rubine's algorithm has a set of pre-processing such as a resampling, rotation, rescaling and translation. Fig. 3 (left) illustrates the set of parameters for that algorithm. Fig. 3 (right) illustrates the parameters selection for the Levenshtein's algorithm [29]. We observe that we may select any of the standard data pre-processing, but may also provide a set of extra parameters such as the possibility to recognize multi-stroke gestures.

In addition to the set of parameters to be used for each algorithm, the user is also free to specify what kind of shapes to consider for the test. As we mentioned previously, gesture recognition algorithms are often designed for recognizing a specific kind of gesture. Fig. 4 (left) shows a capture of the dialog box allowing specifying the type of gestures to be considered. The user has the possibility to select or exclude a complete group or to specify individually between all the groups items.

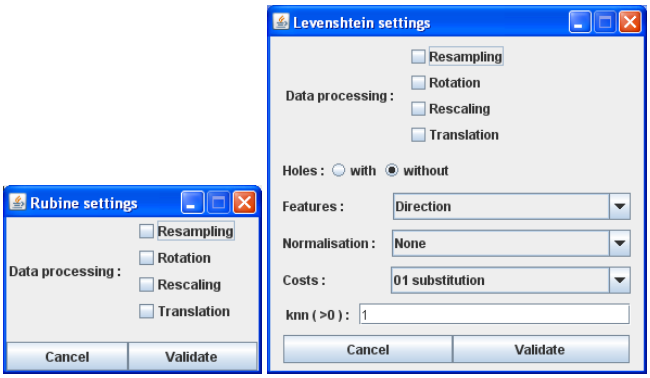


Figure 3. Rubine and Levenshtein settings.

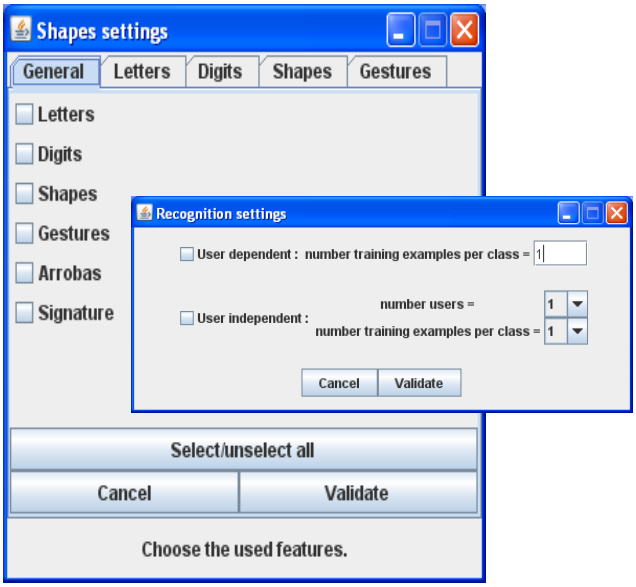


Figure 4. Shapes and recognition settings.

The training is one of the most important aspects that differentiate an algorithm from another. In order to enable the tester to highlight the impact of this aspect, the application allows to train the algorithm with a mix of all the gestures provided by the previous users, but also to train the system with only a subset of the participants. Fig. 4 (right) illustrates the possibility to proceed to a user-dependent testing (i.e. by including training examples entered by the person having drawn the testing example) of the application, but also the number of users to be used and the number of samples for each class of gesture.

D. Comparison between the algorithms

The comparative study of this subsection is divided into four parts: an overview of the algorithms selected, the results for the algorithms in their initial and their modified versions, and a summary of the lessons learned.

1) Algorithms selection: As mentioned before, the platform is extensible and may be adapted to include other algorithms. As a first implementation, we selected four different approaches: Rubine [22], One Dollar [9], Levenshtein [28] and stochastic Levenshtein [29]. These techniques are different (except for the two algorithms based on the Levenshtein distance) and cover many aspects than can be taken into account when proceeding to shape recognition [1]. All these four algorithms can be trained in a similar way. Other algorithm like Cali [5], are very powerful for geometrical shapes recognition and are trainable for new gesture.

a) Levenshtein's algorithm is based on the edit distance between two strings thanks to a measure of their dissimilarity. The idea behind the distance [29,31] is to transform one string into another string using the basic character wise operations delete, insert and replace. The minimal number of these operations for the transformation of one string into another one is called the edit distance or Levenshtein distance. The smaller is the minimal number of needed edit operations for a transformation from string A to string B, the smaller is the distance between these strings.

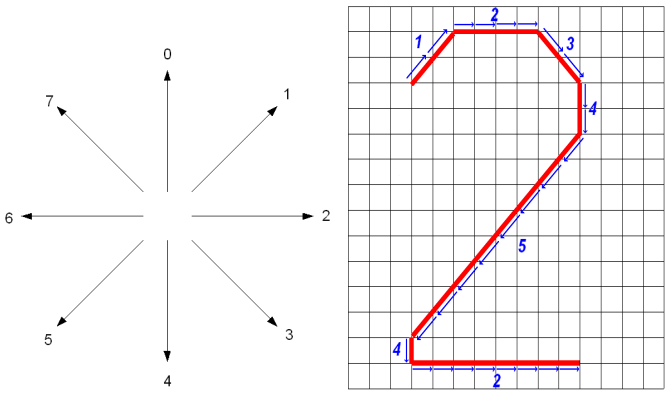


Figure 5. Square grid quantization of freehand shapes.

The features to be extracted from the raw data are based on the idea described in [27]. The drawing plane is superimposed with a grid and the freehand drawing input is quantized with respect to the grid nodes (Fig. 5). Each grid node has 8 adjacent grid nodes and for each pair of adjacent nodes one out of 8 directions can be given. So, from the sequence of successive

grid nodes, a sequence of directions can be derived. This sequence can be coded using an alphabet 0-7, each value representing one direction. Thus when comparing a sequence representing the gesture to recognize with templates present in the training set, we look for the template with the smallest edit distance.

b) Stochastic Levenshtein's algorithm [30] is very close to Levenshtein's algorithm, which is based on the number of suppressions, insertions and substitutions to transform one chain of characters into another. These operations are modeled with a fixed cost for each operation. An intuitive improvement consists in automatically learning these costs in such a way that the precision of the comparisons is increased. This is the purpose of the stochastic Levenshtein's algorithm that is based on a probabilistic model to achieve this. The procedure to recognize a gesture is thus similar to the classic version of Levenshtein, the gestures are firstly transformed into sequences of characters and then compared with the gestures of the training set to find its corresponding class. The only difference is the use of a stochastic approach during the strings comparison. The main goal behind the selection of stochastic Levenshtein is the evaluation of the performances of a statistical-based algorithm. The advantage is its quite low time cost while most of the statistical-based algorithms are very expensive. Furthermore it represents a natural improvement of the traditional Levenshtein algorithm and is interesting to compare with.

c) Rubine's algorithm [22] is based on geometrical and temporal characterization of the gestures so as to compute a vector of features for each class represented in the training set (by considering elements of the training set of the same class). From these features vectors, weights are estimated for each of the classes. A score for each available class in the training set is computed for a new gesture to be recognized by determining its vector of features, which is then combined with the different weights of the classes via an inner product. The class attributed to the gesture is the one giving the best score. This algorithm is well known in gesture recognition: its simplicity and its good performances make it an obvious choice.

d) One Dollar's algorithm [9] is an algorithm only based on 2D template matching. Its working is very simple: each gesture is initially pre-processed in four steps. The first one is a resampling to have the same number of points for all gestures; the second one consists in a rotation following a certain angle computed such as the shapes are oriented in a similar way; the third step is a rescaling of the gestures to give them the same scale; the final step is a translation of the gestures such that their centroïds match the origin. Once this pre-processing is realized, a simple point-to-point Euclidean distance is computed between the set of training examples and the gesture to be recognized. The class attributed to the new gesture is the class of the closest training example. One Dollar is a new, very simple and cheap algorithm for which very good performances are expected. It is thus very interesting to observe how it behaves with regards to the other approaches.

2) Results for the algorithms in their initial version: The experimental study was carried out according to three criteria. *(i)* The impact of the training type was firstly evaluated following three different ways. The first one is a training made dependently on the user, meaning that we took one user at a time to train the algorithms with a subset of its examples and test them with its other examples not used for the training. The two other ways are user-independent: examples of user tester are not considered for training phase. 10 different users were considered in this study and for the first user-independent training, one of them was used to train the algorithms while the examples of the 9 others were used for the tests. The second user-independent training is a test on the examples of one user when the algorithms were trained with the examples of the 9 other users. *(ii)* The second criterion has considered the type of gesture to be recognized such as letters, digits, action gestures, and geometrical shapes. *(iii)* The last test was aimed at comparing the algorithm's performances.

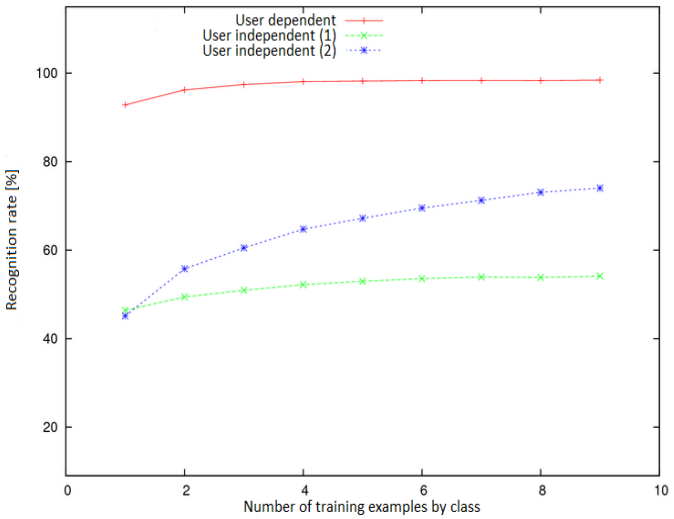


Figure 6. Variation of the training types for Levenshtein's algorithm on digits.

Due to space restriction, only a subset of the analyses is presented in this part, but it gives an overview of the general tendencies. All figures graphically depicting the results are accessible at https://sites.google.com/site/comparativeevaluation/results.

Without going here into the details, the results were obtained through a kind of cross-validation among the examples of the different users. They are provided in term of the average recognition rate on all the users (Y-coordinate) for each number of training examples tested (X-coordinate). Fig. 6 illustrates the results for Levenshtein's algorithm applied on digits with respect to the number of samples. It turns out that a training made dependently on the user provides the best results. However, a preliminary training is not always possible. In this case we can see that the diversity of the trainers is beneficial.

An additional interesting observation is that diversity gives better results in term of absolute performance but also in term of progression as the curve grows unceasingly. It can be explained by the fact that providing examples from a larger set of users increases the probability to find a user with a similar style of writing than the tester and thus find examples that fit for the good class.

For the variation of gesture types, Fig. 7 shows the results for Rubine's algorithm with user-independent training when we vary the types of gesture. This figure illustrates the general tendency. On one hand when a user-dependent training is applied (not represented here), the digits class presents the best recognition rate, followed by the letters, the geometric shapes and the actions. This result can be explained: the digits or the letters are more different among them compared to the geometrical shapes or the action gestures. On the other hand when a user-independent training is used, action gestures provide the best results as they show a lower variance across users. An increasing number of items in a family of classes decreases the performance for this family of classes: letters and digits are two similar families of classes but digits provide better recognition rates because they contain 10 items vs. 26.

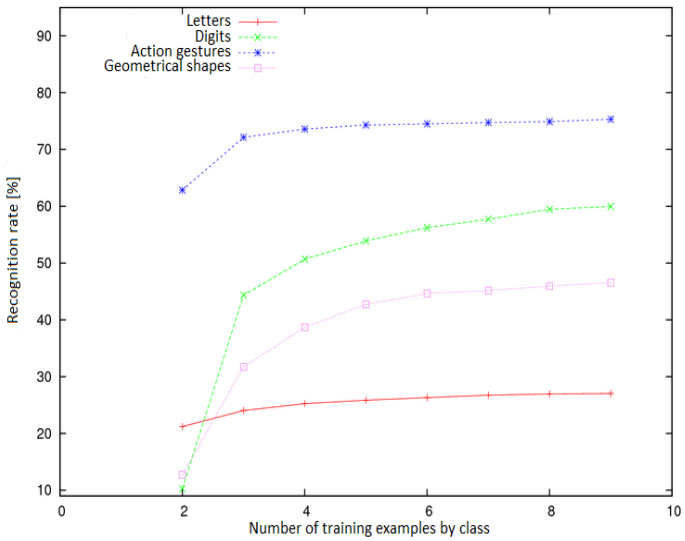


Figure 7. Variation of the gesture types for Rubine's algorithm with user-independent training.

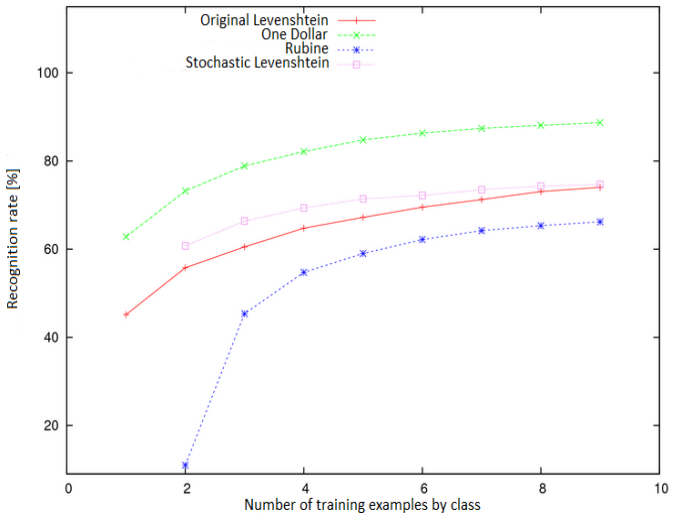


Figure 8. Variation of the algorithms for digits with user-independent training.

For the variation of algorithms, Fig. 8 shows the results for the different algorithms applied on digits with user-independent training. We observed that in most situations, One Dollar provides the best results. This can be explained by the extreme simplicity of the algorithm, which needs only a few numbers of examples to reach its highest performances. Another advantage of One Dollar is its high speed. On the other hand One Dollar's performances are very poor when it comes to recognize gestures that take their difference in geometrical features like action gestures.

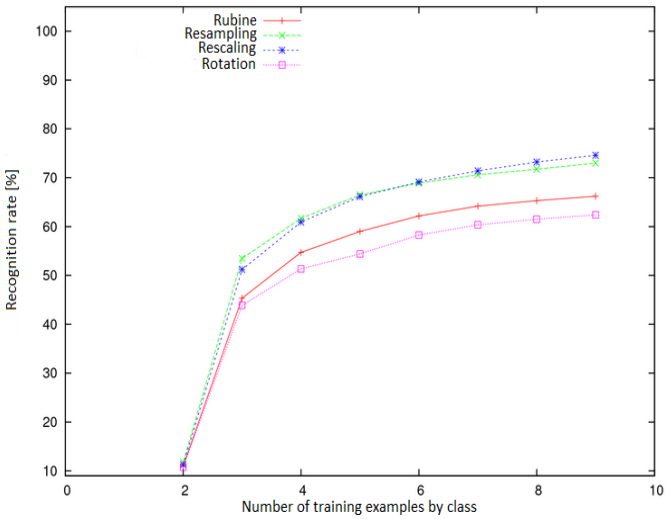


Figure 9. Performances of Rubine's extensions.

3) Results for the algorithms in their modified version: All the results presented previously were based on the initial version of the algorithms. Then, based on our observation, we tried to improve them in different ways. The following is a presentation of the improvements we considered and their effects on the performances. The results presented here are those obtained with a user-independent training with few trainers, since the user-dependent training generates curves (not shown here) too close and do not present much interest for most of the situation. For Rubine's algorithm, we tried to pre-process the data before applying the algorithm. The modifications of the data tested are those used for One Dollar: resampling, rotation, rescaling.

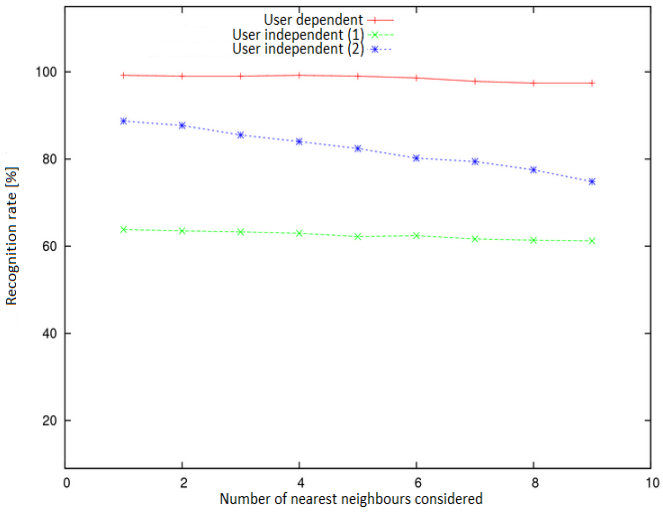


Figure 10. Performances of One Dollar's extensions.

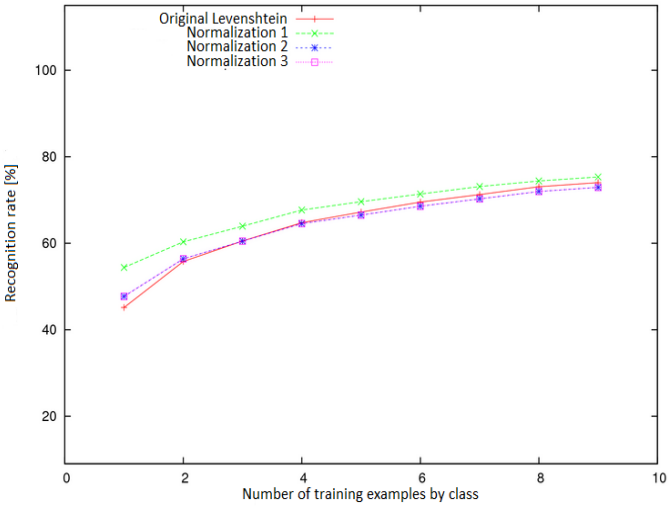


Figure 11. Performances of Levenshtein's normalizations.

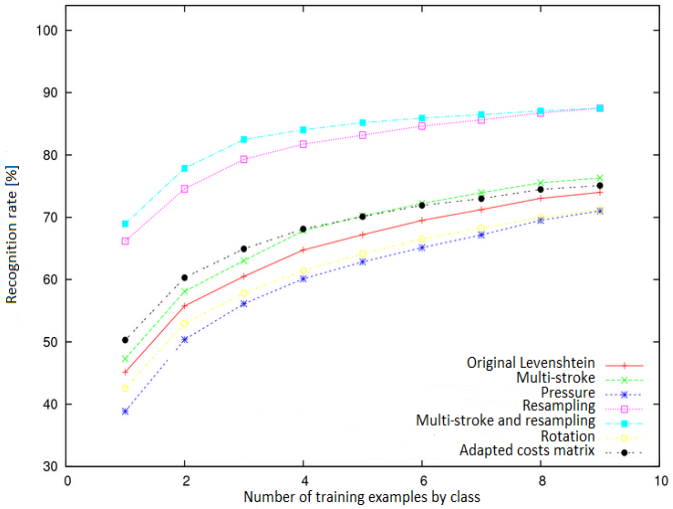


Figure 12. Performances of Levenshtein's extensions.

The resampling and the rescaling are improved (Fig. 9) while the rotation degrades the performances. For One Dollar's algorithm, we performed a test by growing the number of nearest neighbors. As mentioned before, One Dollar's algorithm geometrically transforms the gestures in four steps and then computes the closest class following a Euclidean distance. This last step is called the nearest neighbor. We tried to test the algorithm by taking into account a larger number of nearest neighbors and attribute the class mainly represented in the set of nearest neighbors considered. It can be observed in Fig. 10 that it decreases the performance. It is not really intuitive but the second, third, etc neighbor tends to be less well classified and causes decreasing performances. It is especially true for a user-independent training with a large number of trainers since it is much easier to find a training example sufficiently close to the test example but belonging to a wrong class. For Levenshtein and stochastic Levenshtein's algorithms, several extensions were tested that can be sorted in three categories. First of all, we tested the effect of changing the number of nearest neighbors as we did for One Dollar. We do not show the graphs here but the conclusions are roughly the same as it leads to lower performances. Secondly, we tested normalized distances. Instead of estimating the similarity degree between two chains of character with the classical Levenshtein algorithm, we tested three normalizations of this distance [32]:

- Normalization 1: $\frac{dL}{\max(|A|,|B|)}$
- Normalization 2: $\frac{dL}{|A|+|B|}$
- Normalization 3: $\frac{2*dL}{|A|+|B|+dL}$

Where *dL* is the initial distance of Levenshtein and |A| and |B| are the lengths of the two chains of characters. The first normalization brings improvements in comparison with the initial version while the two others are not really helpful (Fig. 11).

The last illustration (Fig. 12) presents other extensions. The first one is the modification of the chains of characters. Instead of only using the directions to build them, we included more information like pressure, inclination or orientation of the pen. None of these tests were concluding for the traditional Levenshtein's algorithm. But, this last modification can improve the stochastic Levenshtein's algorithm if a sufficient number of training examples is provided (not shown here). As we did for Rubine, we evaluated the impact of a pre-processing of the data before applying the Levenshtein's algorithms. If a reorientation of the shapes decreases the performances for both algorithms, the resampling offers real improvements. Translation as rescaling does not change anything.

Another extension consisted in taking account of the multi-strokes gestures. Initially, two points around a gap in a gesture were treated like two successive points of a single stroke gesture. To model the gap, we added a special character when building the chain of characters representing the gesture. The number of special characters added is function of the length of the gap. This extension improves both the traditional Levenshtein's algorithm and its stochastic version. We then tried to combine it with the resampling and we obtained our best improvement.

Finally, we tried another cost matrix for the Levenshtein's algorithm. In the traditional version of Levenshtein's algorithm, the costs for suppression and insertion were 1, and substitute a character to a same character costed 0 while substitute a character to another one costed 1. This is still the case for suppression and insertion but the new costs matrix modifies them for the substitutions. According to the matrix, the substitution between two same characters remains 0 but the cost of the substitution between two different characters is function of the "distance" between the two characters. For example, the cost between a 6 (left direction) and a 2 (right direction) is more important than the cost between a 6 (left direction) and a 0 (up direction) because they are two opposite directions.

4) Lessons learned: Three different lines have been followed during the study of the algorithms in their initial

version. The first one concerns the training types for which the results are unsurprisingly better for the user-dependent training method. However, such conditions are not always satisfied and in this case, introducing maximum user variability in the learning database increases the recognition rate. The second line deals with the different families of classes. For a user-dependent training, letters and even more digits give the best results while the same conclusion can be drawn for the action gestures for user-independent training. Indeed, compared to each other, action gestures offer more similarity and are thus more similar when the user changes while letters or digits hold more user information.

Finally the last line analyses the algorithms themselves. The observed results matched the expected ones. The constraints about deployment and time conducted us to select only a small number of training examples (9) by family of classes. In this configuration, One Dollar offers the best results (except for the action gestures that are too much dependent of the rotation), even if the stochastic Levenshtein also obtains good results. However, it requires much more training examples to give its best. The advantage of One Dollar is increased by its speed, similar to the speed of Rubine or Levenshtein, but about 1000 times faster than stochastic Levenshtein (speed graphics not shown here).

Improved algorithms have been tested too. Again, because of time and deployment constraints, exhaustive investigation was not possible. We thus made some choices and it appears that the real big improvement are obtained for the Levenshtein and stochastic Levenshtein algorithms by applying a resampling as pre-processing on the gestures. Taking the gaps into account still improves a little bit the performances. The other extensions tested do not represent real improvement or may even decrease recognition rates.

TABLE I. ORIGINAL ALGORITHMS SUMMARY

	User-dep.	User-indep. 1	User-indep. 2
Digits	$1	$1	$1
Letters	$1	$1	$1
Actions	Sto. Lev.	Rub.	Sto. Lev.
Shapes	Rub.	Rub.	$1

Table I summarizes the different performances for the algorithms in their initial versions by highlighting the best algorithm for each situation.Table II gives a performance measure for each modification of the algorithms in comparison with their initial version: '—' for hard decreasing; '-' for decreasing; '=' for similarity; '+' for improvement; '++' for hard improvement; 'X' if not applicable.

The two tables only represent a summary and a general tendency for each feature, under certain circumstances (few number of examples by class). Moreover, it only takes recognition rates performances into account. The training and recognition time may be an important consideration too. Both are similar for Rubine, OneDollar and original Levenshtein's algorithms while they are both drastically increased for stochastic Levenshtein's algorithm.

TABLE II. MODIFIED ALGORITHMS SUMMARY

	Rub.	$1	Lev.	Sto. Lev.
Resampling	+	X	++	++
Rescaling	+	X	X	X
Rotation	-	X	-	-
k*NN*	X	-	-	-
Multi-stroke	X	X	+	=
Multi-stroke + resampling	X	X	++	++
Pressure	X	X	--	-
Normalization 1	X	X	+	-
Normalization 2	X	X	=	-
Normalization 3	X	X	=	-
Adapted costs matrix	X	X	+	X

IV. ENGINEERING OF GESTURE INTERACTION INTO INTERACTIVE SYSTEMS

This section is aimed at describing the way we want to allow a designer integrating gesture interaction in an interface. For this purpose, a case study is firstly presented in order to illustrate the second part explaining the way the link between the gesture recognition and the interface modeling is accomplished.

Figure 13. Initial version of Woke.be UI.

A. Case study

1) Initial version: The Asian food website (http://www.woke.be) enables customers to compose and order their own "Wokbox", a box containing Asian food. The UI is relying on a point-and-click interaction technique (Fig. 13): the user moves the pointer onto any wished ingredient (e.g., vegetables and proteins, starchy food, sauce) presented through multiple screens and click on it in order to increment its quantity. Each ingredient is then added in a pan. In order to cancel any

ingredient or to reduce its quantity, one has to select the ingredient again and to click again on a context-sensitive menu. The whole interaction subsumes several operations, like cursor movement, menu opening, menu item selection, and screen bidirectional navigation.

Another drawback is the lack of visual feedback when too much ingredients are selected. In this case, it becomes difficult to distinguish every ingredient in the woke since they may be hidden by other ones. Moreover, one has to count the occurrences for a same ingredient in order to know how much times it has been selected since no information about this number is proposed.

2) Redesigned version: The Woke UI would be redesigned with pen-based interaction so that there is only one screen presenting the food under order, equipped with gestures to represent quantities, ordering, etc. The following gesture set has been defined: the ten digits for setting the quantity of each ingredient by sketching it on the wanted ingredient (0 for canceling the ingredient), the "Right" (→) and "Left" (←) action gestures to undo or redo the last action, the "Up" (↑) and "Down" (↓) action gestures to increase or decrease any ingredient by one unit, and the triangle (Δ) to cancel the whole order or an alpha (α) to withdraw anyway an ingredient. A gesture from an ingredient to the pan would also add this ingredient.

In order to avoid the woke being overwhelmed by too many items, only one occurrence of each ingredient is depicted in the woke. A digit indicating the number of occurrences is represented on each selected ingredient.

Figure 14. Redesigned version of Woke.be UI.

B. *Selection of the gesture recognition mechanism*

The previous section has extensively discussed the results of experimental studies for addressing the first requirement, i.e. selection of an appropriate algorithm depending on various parameters. Although the study is not exhaustive, it allows considering four different algorithms with extensions, according to variations of dataset and types of training. A developer willing to create a user interface including pen-based gesture recognition may rely on the results in order to select the appropriate recognition mechanism among the proposed possibilities.

In the Woke case study, two kinds of gestures are involved: digits and action gestures. Such a system is only occasionally manipulated by users during a limited time, asking them training the system would not be suitable. A user-independent training method is then selected. Moreover, stochastic Levenshtein's algorithm is too slow and should be discarded.

As the dataset is already built with the required gestures, many training examples are available the first method of user-independent training method is selected. In this case, it can be learned from Table I that either $1 or Rubine algorithms give best performances depending on the type of gesture considered ($1 for digits, Rubine for action gestures). By studying more carefully the graphs, one can see that the performances for Rubine's algorithms on digits remain good while the performances of $1's algorithm on action gestures are very bad. Rubine is then selected to be used in the woke system. A resampling and a rescaling may even be applied on gesture before recognition as Table II shows the benefits on performances.

C. *Integration into an engineering framework*

In order to address the fourth requirement, i.e., integration in UI development, different alternatives have been investigated, reviewed, and tested against the following criteria: ability to incorporate multiple gesture recognition algorithms, ability to encapsulate a module for algorithm selection based on parameters, openness, and extensibility. From the various candidates (among them were [15,20]), the SKEMMI [33] UI editor was selected since it allows integrating and connecting different components that ensure multimodal interaction with its own functionalities and parameters, thus supporting device independence, flexibility, and reusability. A SKEMMI *Component* consists of a bundled piece of software that provides a set of services/functionalities (*Facets*), which include input device drivers, signal-processing algorithms, fusion, fission, network communications, and graphical UIs [18]. These facets have well defined input/output interfaces called *Pins*. A pin can be a *Sink* or *Source* (used to receive data), or a *Callback* used to send data (and to import external functionalities into a facet). These main entities are illustrated in Fig. 15.

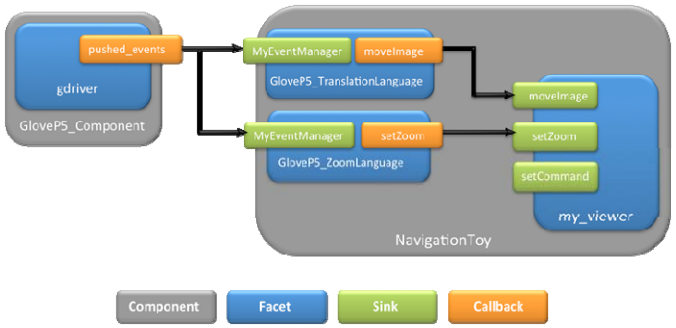


Figure 15. Main entities of Skemmi.

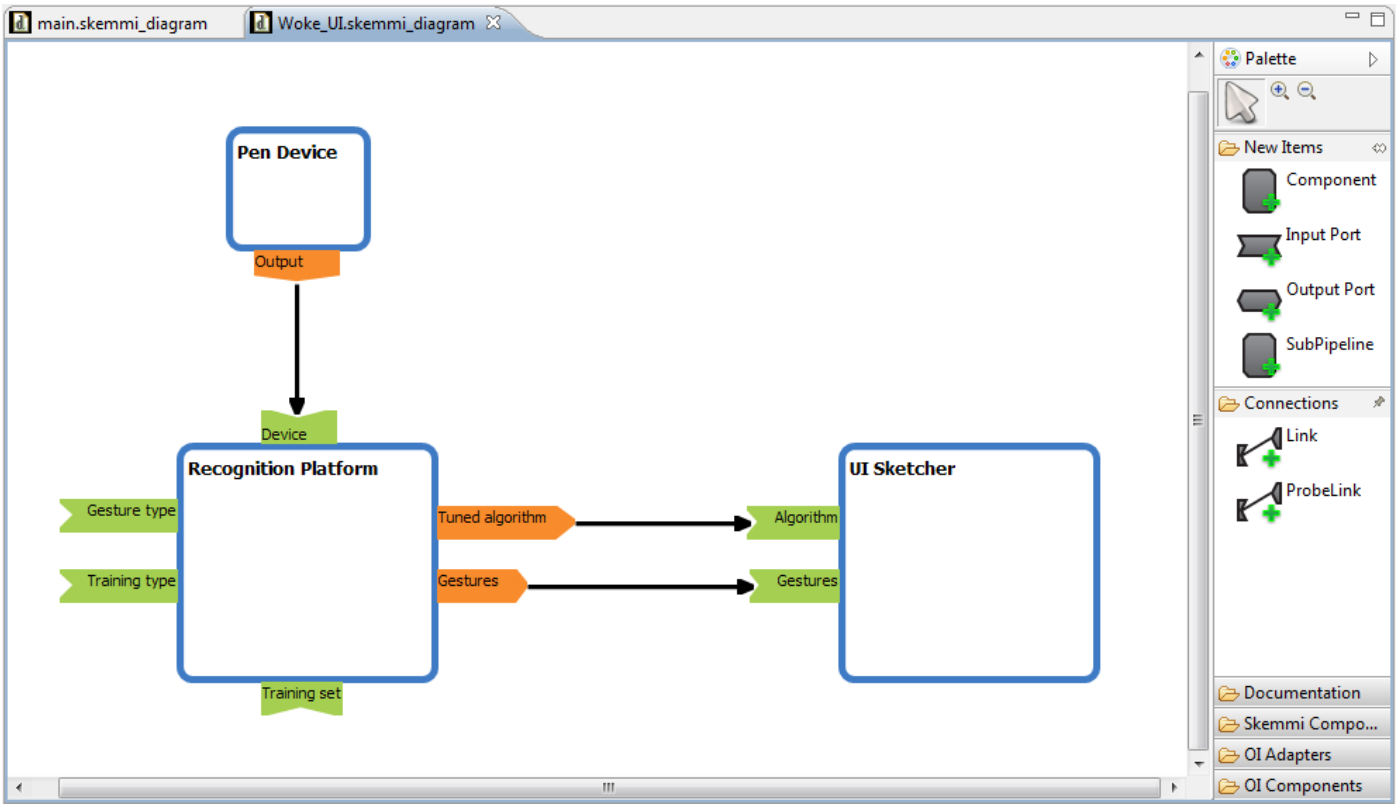


Figure 16. Skemmi integration.

In order to integrate the approach into UI development, it was decided to develop a SKEMMI component having sinks (i.e., the type of gestures selected, the training gesture set, the training type, with or without examples provided by the user) and having one source, a pen device. Note that different pointing devices are supported in SKEMMI, thus supporting some device independence. Considering our case study, additional parameters include the digits, the four action gestures, the alpha, and the triangle as gesture types, the database already present in the system as training gesture set, a user-independent training as training type with few variability among users as training type (it is possible to ask the user to provide its own training examples for this kind of application). The component then provides two callbacks: the selected recognition algorithm with fine-tuning (see Table I and experimental results) and the recognized gestures to be integrated with other components (like in Fig. 16). In our case, Rubine (with gesture resampling and rescaling) would be the most suitable algorithm according to the previous discussion. Additional information would be determined in this component, like the different parameters of Rubine: the goal is to automatize the process of algorithms and features selection followed this discussion.

The recognition callbacks may then be connected to a UI Sketcher module allowing the designer to sketch an interface by using the tuned algorithm and gestures determined by the platform.

V. CONCLUSION

Pen-based interaction certainly brings some advantages for UIs and may enact the user experience. However, incorporating pen-based gestures into an existing UI remains a challenge for a designer due to several problems that require some knowledge of the gesture recognition field. Throughout this paper, we have presented a method enabling a designer to incorporate gesture interaction in UI development without requiring this knowledge. The first part presented a gesture recognition platform to be used to compare and identify the most suitable algorithm and settings according to the designer requirements, thus addressing the requirement of algorithm selection. All the results from these experimental studies have been incorporated in a SKEMMI, a UI editor for assembling components for interaction techniques, as explained in the second part, thus achieving device independence: SKEMMI is itself device independent since it provides designers with an abstraction of the underlying device and algorithm mechanisms. For the designer, there is no need to master the recognition algorithms since their respective performance is included in the selection mechanism. In addition, it supports flexibility and reusability since the same design could be ported to another application or modified within the editor. Informal feedback from testers and designers shows the following improvements to be brought in the near future: need for explaining the algorithm selection on-demand, need for WISYWIG representation of the resulting UI with pen-based gestures, and need for a prototyping tool.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of the ITEA2-Call3-2008026 USIXML (User Interface extensible Markup Language) European project and its support by Région Wallonne, Direction générale opérationnelle de l'Economie, de l'Emploi et de la Recherche (DGO6).

REFERENCES

[1] Artières, T., Marukatat, S., Gallinari, P., Online handwritten shape recognition using segmental hidden Markov models. *IEEE Trans. on Pattern Analysis and Machine Intelligence 29*, 2 (Feb. 2007), pp. 205–217.

[2] Briggs, R., Dennis, A., Beck, B., and Nunamaker, J. Whither the pen-based interface? *Journal of Management Information Systems 9*, 3 (Winter 1992–93), pp. 71–90.

[3] Long, A.C.J., Landay, J.A., and Rowe, L.A.. Implications for a gesture design tool. In: *Proc. of the ACM Conf. on Human factors in computing systems CHI'99* (Pittsburgh, May 15-20, 1999). ACM Press, New York (1999), pp. 40–47.

[4] Appert, C. and Zhai, Sh. Using Strokes as Command Shortcuts: Cognitive Benefits and Toolkit Support. In: *Proc. of CHI'2009* (Boston, 4-9 April 2009). ACM, pp. 2289-2298.

[5] Fonseca, M. J., Pimentel, C., and Jorge, J.A. CALI, An online scribble recognizer for calligraphic interfaces. In *Proc. of the 2002 AAAI Spring Symposium on Sketch Understanding*, (2002), pp. 51–58.

[6] Wolf, C., Can people use gesture commands? *ACM SIGCHI Bulletin 18*, 2 (1986) 73–74.

[7] Hinckley, K., Ramos, G., Guimbretière, F., Baudish, P., and Smith, M. Stitching: pen gestures that span multiple displays. In: *Proc. of AVI'04*. ACM Press, NY (2004), pp. 23-31.

[8] Wilhelm, M., Roscher, D., Blumendorf, M., and Albayrak, S. A Trajectory-Based Approach for Device Independent Gesture Recognition in Multimodal User Interface. In: *Proc. of HAID' 2010*, pp. 197–206.

[9] Wobbrock, J.O., Wilson A.D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In: *Proc. of UIST'07* (New York, 2007). ACM Press, New York (2007), pp. 159–168.

[10] Blagojevic, R., Schmieder, P., and Plimmer, B. Towards a Toolkit for the Development and Evaluation of Sketch Recognition Techniques. In: *Proc. of ACM IUI'09 Workshop on Sketch Recognition* (Sanibel Island, February 8, 2009).

[11] Goh, W.L., Mital D.P., and Babri, H.A. An artificial neural network approach to handwriting recognition. In: *Proc. of 1st Int. Conf. on Knowledge-Based Intelligent Electronic Systems KES'97* (Adelaide, May 1997). Vol. 1 (1997), pp. 132–136.

[12] Patel, R., Plimmer, B., Grundy, J., and Ihaka, R. Ink Features for Diagram Recognition. In: *Proc. of 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling SBIM'07* (Riverside, 2007). ACM Press, NY (2007), pp. 131–138.

[13] Paulson, B. and Hammond, T. PaleoSketch: Accurate primitive sketch recognition and beautification. In: *Proc. of IUI'2008*. ACM Press, New York (2008), pp. 1-10.

[14] Paulson, B., Wolin, A., Johnston, J., and Hammond, T. SOUSA: Sketch-based Online User Study Applet. In: *Proc. of SBIM'08* (Annecy, 2008). ACM Press (2008), pp. 81–88.

[15] Schmieder, P., Plimmer, B., and Blagojevic, R. Automatic Evaluation of Sketch Recognizers. In: *Proc. of 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling SBIM'09* (New Orleans, August 1-2, 2009). ACM Press, New York (2009), pp. 85–92.

[16] Cao, X. and Zhai, S. Modeling human performance of pen stroke gestures. In: *Proc. of ACM Int. Conf. on Human factors in Computing Systems CHI'2007*, pp. 1495–1504.

[17] Frankish, C., Hull, R., and Morgan, P. Recognition accuracy and user acceptance of pen interfaces. In: *Proc. of CHI'95*. ACM Press, New York (1995), pp. 503-510.

[18] Long, A.C.J., Landay, J.A., Rowe, L.A., and J. Michiels. Visual similarity of pen gestures. In*: Proc. of the ACM Conf. on Human factors in computing systems CHI'99* (The Hague, April 1-6, 2000). ACM Press, New York (2000), pp. 360–367.

[19] Lyons, K., Brashear, H., Westeyn, T., Kim, J.S., and Starner, T. GART: The Gesture and Activity Recognition Toolkit. In: *Proc. of HCI International 2007* (Beijing, July 22-27, 2007). LNCS, Vol. 4552. Springer, Berlin (2007), pp. 718–727.

[20] Signer, B., Kurmann, U., and Norrie, M. C. iGesture: A General Gesture Recognition Framework. In: *Proc. of 9th Int. Conf. on Document Analysis and Recognition ICDAR'2007* (Curitiba, September 2007). IEEE Computer Society Press, Los Alamitos (2007), pp. 954-958

[21] Long, A.C.J. quill: A Gesture Design Tool for Pen-based User Interfaces. PhD Thesis, Univ. of California at Berkeley, 2001.

[22] Rubine, D. Specifying gestures by example. *SIGGRAPH Computer Graphics 25*, 4 (July 1991), 329–337.

[23] Hong, J. and Landay, J., Satin: a toolkit for informal ink-based applications. In: *Proc. ACM Symposium on User interface software and technology UIST'00* (San Diego, November 5-8, 2000). ACM Press, New York (2000), pp. 63–72.

[24] Okumur, D., Uchida, S., and Sakoe, H. An HMM implementation for on-line handwriting recognition based on pen coordinate feature and pen-direction feature. In: *Proc. of ICDAR'05* (Aug. 29-Sept. 1, 1995). IEEE Comp. Soc. (1995), pp. 26–30.

[25] Unipen, http://unipen.org. Last visited on Nov. 5th, 2010.

[26] The UC Irvine machine learning repository. Accessible at http://archive.ics.uci.edu/ml/. Last visited on Nov. 5th, 2010.

[27] Llorens, D., *et al.* The Ujipenchars database: a pen based database of isolated handwritten characters. In: *Proc. of the 6th Int. Language Resources and Evaluation LREC'08* (Marrakech, May 28-30, 2008). European Language Resources Association (ELRA), 2008.

[28] Mackenzie, I.S. and Zhang, S.X. The immediate usability of Graffiti. In: *Proc. of the Conf. on Graphics Interface GI'97* (Kelowna, May 21-23, 1997). Canadian Human-Computer Interaction Society (2009), pp. 129–137.

[29] Coyette, A., Schimke, S., Vanderdonckt, J., and Vielhauer, C. Trainable Sketch Recognizer for Graphical User Interface Design. In: *Proc. of INTERACT'2007* (Rio de Janeiro, September 10-14, 2007). LNCS, Vol. 4662. Springer, pp. 123–134.

[30] Oncina, J. and Sebban, M. Learning stochastic edit distance: Application in handwritten character recognition. *Pattern recognition 39* (2006), pp. 1575–1587.

[31] Soukoref, R.W. and Mackenzie, I.S. Measuring errors in text entry tasks: An application of the Levenshtein string edit distance. In: *Proc. of EA-CHI'2001*. ACM Press, pp. 319–320.

[32] Yujian, L. and Bo, L., A normalized Levenshtein distance metric. *IEEE Trans. on Pattern Analysis and Machine Intelligence 29* (2007), pp. 1091–1095.

[33] Lawson, J.-Y., Al-Akkad A.-A., Vanderdonckt, J., and Macq, B. An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-The-Shelf Heterogeneous Components. In: *Proc. of EICS'2009* (Pittsburgh, July 15-17, 2009). ACM Press, New York (2009), pp. 245–253.