Application for the search of text in handwritten notes in mobile devices

Jose Miguel del Caño Masip

Abstract—In this document you will find information about the creation of the handwritten search application. This project's main objective is developing an application for searching words from a document filled with pen strokes. Nowadays there are applications allowing users to load handwritten notes, save and edit drawings, and much more. However, none of them implement the much-needed functionality of searching text from handwritten notes. It is right here where my application comes in handy offering a unique solution. This project will cover many aspects such as the basics of developing an application, learning how the Samsung Pen SDK works and explaining all the background process that is needed to create the search functionality. In addition, it will describe the difficulties that I have found during the process of developing the application.

Index Terms— Software Development Kit: The Software Development Kit, also known as SDK, is typically a set of software development tools that allows the creation of applications for a certain software.

Activity: The Activity class serves as the entry point for an app's interaction with the user, providing the window in which the app draws its UI. This window typically fills the screen but may be smaller than the screen and float on top of other windows. The Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

Layout: It contains the View part in the Model-View-Controller paradigm of an application. Each item (buttons, menus, toolbars) is defined in the layout and then referred inside an activity via ids.

Hidden Markov Model: Also known as HMM, is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. These models are known for their application in temporal pattern recognition such as speech or handwriting.

Cosine similarity: Measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

1 Introduction

THROUGHOUT the beginning of the 21st century, many different technologies have been appearing. Specifically, with the modernization of mobile devices and the appearance of smartphones and tablet computers, portable touchscreen devices have been dominating the global market. Some of them started incorporating an additional feature, the electronic pen. Its utilization, more focused towards tablet computers, allowed users to transform touchscreen displays into notebooks. And with it, drawing or taking handwritten notes started to be a reality for casual users.

At the same time, the market was in the need of some applications that could handle these new features.
Despite the creation of many applications such as loading and saving pages with pen input, none of them implemented a search function for handwritten documents. Although it may not seem necessary, there are a lot of cases where this could be really helpful. That was the main reason this application was created. The introduction of this application to the market is expected to cover this yet unsatisfied need.
The main objective of this project was to effectively develop an application that could search any given handwritten input from a document filled with pen strokes.

2 Objectives

This project has three main objectives: establishing target devices, building an application and implementing a search function.

Firstly, this application has been built for Samsung devices that support SPen. Regarding Android SDK restrictions, I want this application to be executable in as many devices as possible.

Secondly, building an application for ANDROID OS hasbeen the next goal. This implies learning the basics of

- *E-mail: josemi.masip@gmail.com*
- *Studies: Computació*
- *Project supervised by: Ernest Valveny (Computació)*
- *Curs 2016/17*

mobile programming and project structure.

Thirdly, creating a search function that based on the similarity from a given word and a set of loaded words, retrieves the words with the highest similarity. In order to do this, I have utilized tecnhiques of computer vision.

3 STATE OF THE ART

First off, there is a text recognition plug-in included in the sample application of the SPen SDK. Despite developers not having access to its code, it can still be utilized. At first, I was willing to include it in my project, but the results were neither great nor accurate. Furthermore, I was not able to access the code, so I decided that spending time on it would be a waste of time. This sample application had a pen mode to draw, and a selection tool to manually select which part the user would like to transform into text. After selecting it, the process was done automatically.

In figure 1, you can see how this plug-in works in the sample application with the two buttons I just explained.

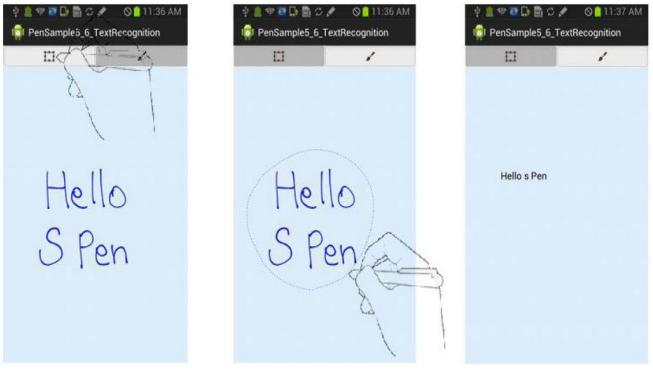

Fig. 1. Example of utilization of the SPen Text recognition Plug in. In the sample application, the user has to first type something, and then select the part that he wants to be transformed into characters.

Apart from this plug-in, there are three projects that were similar to mine. After carefully analyzing them, I could manage to utilize them as guidelines for my project. The titles of the first two projects are *HMM-based On-line Recognition of Handwritten Whiteboard Notes* [1] and *NPen++: An On-Line Handwriting Recognition System* [2]. The concept of these two projects is the same as mine. Starting with a preprocessing phase, then moving to the feature extraction and finally going for the recognition system. That said, it is important to notice that those projects' main goal is to recognize the words whereas in my project I am looking for the similarity between words. That is the reason why Bag of Words may be better than what they utilize in their approaches.

There is a third article, *Word Spotting in Historical Document Collections with Online-Handwritten Queries* [7] that also utilizes the BoW algorithm. In this article, you will find a much deeper and robust approach towards word spotting. Its goal is also looking for similarities between image words, not recognizing words such as the first two articles. However, since this article is too similar, I will just explain the other two to give you different perspectives.

3.1 Preprocessing

These two projects have similar approaches towards preprocessing the data. The HMM project works with text line, breaking them into components and correcting skew and slant. In figure 2, you can find an example showing how this process works.

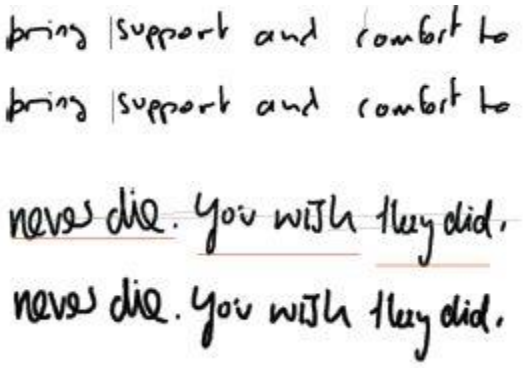

Fig. 2. Examples of the correction of skew and slant introduced in the HMM project.

Finally, it finds the baseline and the corpus line of the text lines. In figure 3, you will find an example of finding the corpus line of a given word.

Fig. 3. Example of corpus baseline introduced in the HMM project.

The NPen++ project also computes the baselines. Then, it normalizes size, rotations, and inclinations. It also interpolates missing points to offer support in situations where the hardware is not capturing all the points. Finally, it goes through three more processes: smoothing, resampling and removing delayed strokes. This last step removes for instance strokes such as the cross of the "t".

3.2 Feature extraction

Both projects share the majority of the features. In my project, I have implemented all the possible features that I could. These features are further elaborated in section 5.2 of this article.

3.3 Recognition system

The recognition system is unique for each project. In the HMM project, Hidden Markov Models are used together with a statistical language model whereas in the NPen++ project utilizes a neural network and the results are way better than the HMM.

4 METHODOLOGIES

In this section, you will find the basics of building an application, an explanation about how do the SPen SDK structures work and how the words have been detected and prepared to be processed.

4.1 Learning the basics to develop an application

This phase splits into two well-differentiated parts: design and develop. Since I have personally spent more time in the development part, I will start there.

Firstly, I had to understand how the flow of an application worked. Understanding the core of the activity-lifecycle of an application allows you to not only develop better applications, but to prevent memory leaks code with more efficiency as well.

To start with, I will explain the two most important stages, which are onCreate and onDestroy. These two callbacks are invoked on different states:

- OnCreate() gets invoked when the activity starts for the first time. Therefore, althought not every state is mandatory, every activity must have its own onCreate() state.
- OnDestroy() is the last call that the activity receives and is called before the activity is destroyed. This callback releases all resources that may not have been released by earlier callbacks such as onStop().

Notice that I am not talking about the application's start or end. These callbacks are attached to one activity meaning that, for instance, every activity will have its own onCreate. In figure 4, you will find an image showing the activity lifecycle of an application including the different stages.

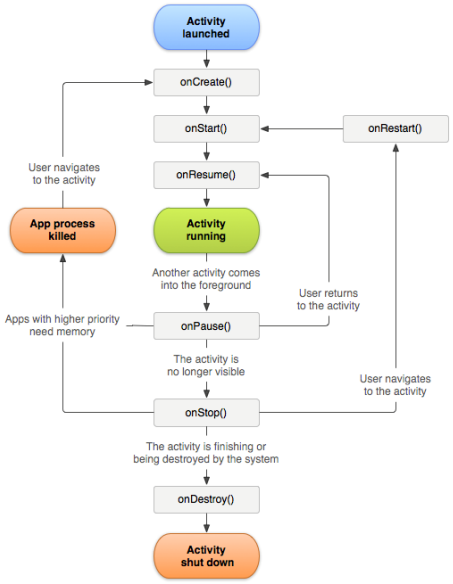

Fig.4. Activity lifecycle of an application showing the main callbacks involved in its execution.

Secondly, there was the Manifest. Every application has its own manifest. This document provides your Android system with essential information about the application. Necessary information that the system requires in order to run the application. For example, the minimum Android SDK or the permissions that this application should have, are included in the manifest.

Thirdly, there was the design phase. Although I have not invested a lot of time into this part, it is worth to mention. In Android, the developer has a lot of options to design and customize its own application utilizing buttons, menus, toolbars and much more. Learning how to create, define, and assign layouts has been also part of the learning process.

4.2 Understanding and working with the SPen SDK

SPen SDK contains a set of software tools and functions that can be utilized to work with the pen. This SDK has a sample project where any developer could find general functionalities to add to their own projects. In my case, the whole application project has been built over this sample. Concretely, it has been built over the load files functionality, which was already implemented.

The main problem I have found while trying to understand the SPen SDK is its really poor API. Most function descriptions are ambiguous and too short, thus leading to many occasions where I had to experiment with the code in order to see what some mehods were truly meant to do. Regardless of this, the first thing I had to understand from the SDK was how Samsung managed the documents in which users draw the data.

On the lowest layer, there is the SpenSurfaceView, the surface that manages everything that involves the utilization of the display. We can attach one SpenPageDoc to this surface. And this SpenPageDoc is part of a SpenNoteDoc. So, SpenNoteDoc does the function of a notebook whereas the SpenPageDoc would be a single page inside the notebook. Despite this, each SpenNoteDoc starts with no pages at all attached upon creation.

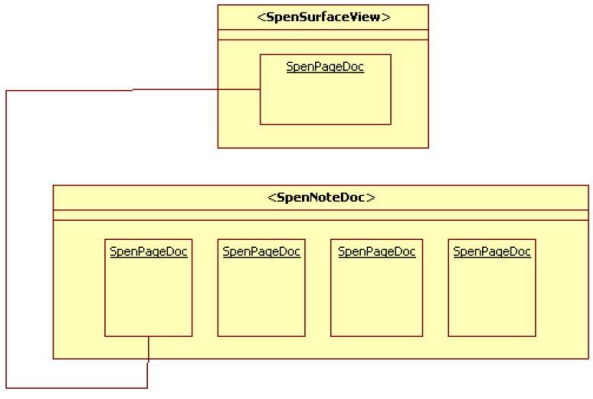

Fig. 5. Structure of the SPen SDK functions that allow developers retrieving data and working with the Samsung Pen.

To sum up, the real information is gathered in the SpenPageDoc attached to the SpenSurfaceView in the moment of drawing or loading a file. This class has a method that returns a list with the information of all the objects contained. The problem is that Samsung manages differently each type of object, and the list returns the super class of all objects, SpenObjectBase which has no useful information at all. To solve this, every SpenObjectBase retrieved from the page was cast into a SpenStrokeObject. This type of object had all the information I needed such as coordinates, timestamps, pressure or tilts. From now on, I could start working with StrokeObjects. The next step was detecting words.

4.3 Detecting words

In order to detect words, we first needed to know what a SpenObjectStroke truly was. Spen creates an object with the information from the moment the pen touches the screen, to the moment the user lifts it. For better understanding of this, check figure 6.

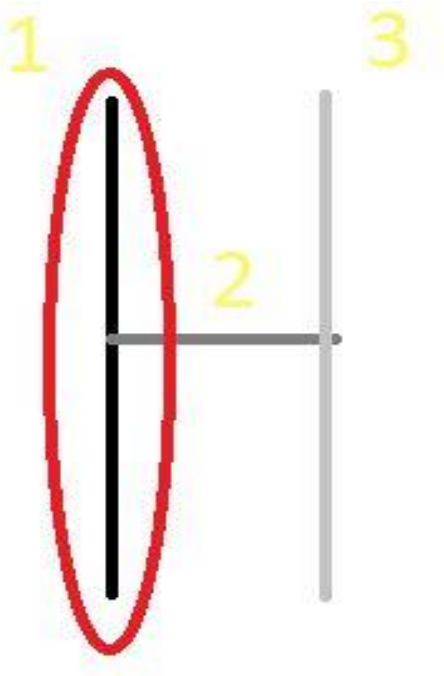

Fig. 6. Detection of objects by the SPen SDK. In the capital letter 'H', you can observe the different objects that are being detected. Samsung pen detecs one object as soon as the user raises the pen after drawing a stroke.

As we can see, the pen detects 3 different objects. 1 and 3 being the vertical lines, and 2 being the middle line that links them. Understanding this was crucial in this project.

The second step was deciding how to correctly detect each word. I have implemented a really basic function that based on two given consecutive objects, it calculates the distance from the maximum X coordinate of the first object to the minimum X coordinate of the second object. Then, I compared that distance from a preset threshold and, if it was bigger, the function established the next object as a new word and ended the current one. Otherwise, it would keep working with the same word. At first, after correctly adjusting the threshold, it worked neatly. But afterward, I had to optimize it because in some cases such as accents, it did not work properly.

The optimization I went for was just for the cases where the distance was bigger than the threshold. Every type of object that you can find in the Spen SDK has a method getRect that returns the rectangle surrounding the stroke. In my optimization, instead of calculating the distance between two given Xs of two different objects, I calculated the distance between the coordinate X of the center of this rectangle of both objects. This way, if it is the case of an accent or something similar, this distance would be minimal. Thus, it would not count as a new word. After testing it, this optimization worked as expected.

5 Recognition system

In this section, you will find explained how does the recognition system work and which parts it contains. Normalization, creation of feature vectors and Bag of Words are its stages. Finally, it is explained what it is done with the information obtained.

5.1 Preprocessing the data

At this point, I had the input data, and I was able to detect different words. So, what was next? Normalizing this data to try to minimize variability between different handwritten styles. To do that, I did mainly two things: size and coordinates.

Firstly, setting every word's height to a preset height. Even though this may seem quite simple, it turned out to be harder than expected because on my first tests, the letters of the same word were stacking one to another after escalating.

Fig. 7. Example of a bad reescalation that was found during the creation of the preprocessing phase. As you can see, letters were stacking not respecting their own space.

In order to prevent this from happening, I had to do the next steps:

1. Transforming each individual object's rectangle contained in a word to the rectangle union of all objects inside that word. In the image below, the red rectangle would represent object number 3 in the "H" image and the green rectangle would represent the same object after the process. The function would have to do the same with every object inside the word. That implies the first two objects of the "H" letter, and the other 4 objects representing the letters "e", "l", "l" and "o".

Fig. 8. Example of a correct reescalation technique. In this example, you can see how the rectangle representing one of the vertical lines of the H gets transformed into the whole word's rectangle. After doing this to all the objects, this rectangle is reescalated into the desired height.

2. Calculating the aspect ratio (AR) of the rectangle. After doing this, we start setting the coordinates of the next rectangle. To set a new rectangle, the function needs 4 coordinates: left, top, right and bottom. Being left and bottom respectively the minimum X and Y of the new rectangle, and right and top the maximum. The key of this step is leaving bottom and left with the same values, then setting the desired preset height, and then calculating the remaining variable "right" based on the next formula:

$$AR = \frac{X}{Y} \rightarrow X = AR * Y$$

3. Utilizing the method of SpenObjectBase setRect. This method needs two parameters. the first is a rectangle object which is just the one we created in step 2. And the second one is a boolean. By setting this boolean to false we allow the function to change the internal data of its rectangle. Consecuently, it automatically escalates the content.

Secondly, the next normalization step was relocating each word to the same place. This would allow me to actually compare relative coordinates rather than real coordinates because all the words compared would start from the same location. Notice that this preprocessing phase involving reescalating and relocating was a background process that did not modify at all the loaded file data. After doing this process of normalization, I fixed a crucial mistake that was returning incorrect words based on where the user typed its stroke. Figure 9 shows this problem.

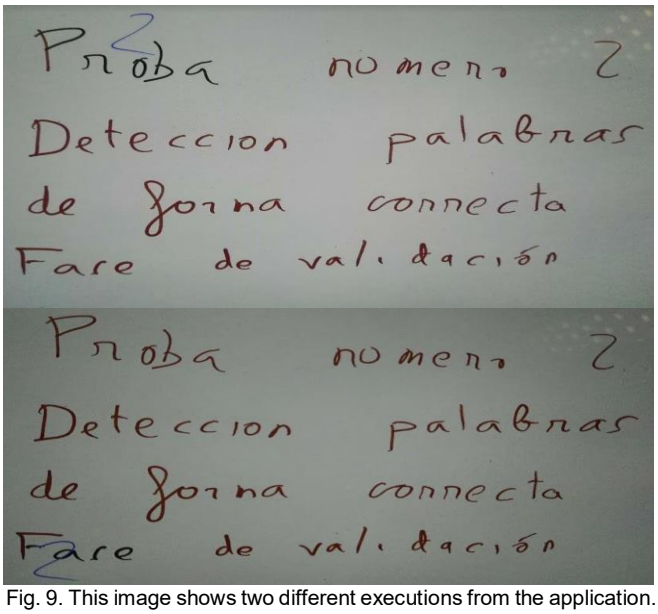

Fig. 9. This image shows two different executions from the application. The black ink represents the word that is returned and matches the word typed by the user. The blue ink represents the word searched.

The last normalization step was changing the Spen pen width to a certain width so that you could better compare the given word with the list of words loaded.

5.2 Creating the feature vectors

The following step is, for each of the points contained in the objects that form the words, calculating a list of features.

Initially, I created a class named FeatureVectorClass that contained each of the features we would have to calculate afterward alongside the classic encapsulation with the "get" and "set" methods. This way, I will then save all the feature vectors as a linked list formed by FeatureVectorClass objects.

I will proceed to describe all the different features that are being calculated:

- **Coordinates:** X and Y coordinates of the point. It gets the coordinate with a straight method of SpenStrokeObject named getPoints().
- **Writing direction:** calculates the writing direction for the x-axis or the y-axis based on a boolean parameter utilizing the next formula:

$$\cos\alpha(t) = \frac{\Delta x(t)}{\Delta s(t)} \qquad \sin\alpha(t) = \frac{\Delta y(t)}{\Delta s(t)}$$

- **Curvature:** calculates the curvature for the x-axis or the y-axis based on a boolean parameter utilizing the next formula:

$$\cos\beta(t) = \cos\alpha(t-1) * \cos\alpha(t+1) + \sin\alpha(t-1) * \sin\alpha(t+1)$$

$$\sin\beta(t) = \cos\alpha(t-1) * \sin\alpha(t+1) - \sin\alpha(t-1) * \cos\alpha(t+1)$$

- **Vicinity aspect:** the aspect of the trajectory in the vicinity of a point is another local feature which characterizes the ratio of height to width of the bounding box containing the preceding and succeeding points of the current point. It is calculated utilizing this equation:

$$A(t) = \frac{2 * \Delta y(t)}{\Delta x(t) + \Delta y(t)} - 1$$

- **Vicinity Slope:** the slope of the straight line joining the first and last point in the vicinity of the current point is described by the cosine of its angle α.
- **Vicinity Curliness:** Curliness is a feature that describes the deviation from a straight line in the vicinity of the current point. It is defined by this equation:

$$C(t) = \frac{L}{\max(\Delta x, \Delta y)} - 2$$

- **Vicinity Linearity:** the average square distance between every point in the vicinity of our current point and the straight line joining the first and last point in this vicinity. Its equation is defined as:

$$L(t) = \frac{1}{N} * \sum_i d_i^2$$

5.3 Bag of Words

At this point, I had everything prepared to start the core of the search function. The algorithm I have utilized is called Bag of Words. This algorithm consists of first extracting a set of features from each of the words based on different characteristics, then creating a histogram for each word, updating them with the frequency of appearance of each feature vector type, and finally comparing similarities. But before updating the histograms each of the feature vectors are transformed into the nearest feature vector type that has been previously calculated. Therefore, the first part of this search function was precalculating all the feature vector types. In order to do this, I have utilized a clustering algorithm once and then generated a file which will be utilized to extract the data.

The algorithm implemented has been the KMeans algorithm. KMeans is a clustering algorithm that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean.

Kmeans has two main phases:

- Assignment step
- Update step

But before further elaborating, there are three things to be understood: what a cluster is, what is its centroid, and an extra phase I would like to talk about, which is the initialization phase.
I have mentioned before that KMeans is a clustering algorithm but, what is a cluster? Clusters are each of the subgroups that are created after the assignment phase. Clusters have two main functionalities: Uniting similar vectors into one group, and isolating them as much as possible from the rest of the groups. In order to be as effective as possible, a cluster looks forward to minimizing the distance between every observation assigned to the cluster and its centroid (inter-class distance) and maximizing the distance between each of its observations and the observations assigned to other clusters (intra-class distance). Each cluster has its own centroid, which is calculated based on the means of all its vectors within the cluster.

Next, I will talk about the method I have utilized to initialize the clusters, which is the Forgy method. This method consists of randomly selecting k observations from the data and utilizing it as the initial means.

After the initialization, this algorithm enters into the iterative sequence, which will last until convergence or reaching a preset number of iterations. So, since the majority of the time is spent on these two steps, it is important to understand them.

Assignment step: in the first stage of the iterative sequence, each observation is assigned to a cluster based on the nearest mean. To calculate this mean, the equation of the Euclidean distance is used. When every observation has been assigned to a cluster, this step ends.

Update step: in the next and final step, after the assignment process the new centroids are calculated based on the new means.

The algorithm has converged when the assignments no longer change. If it never reaches convergence, it has to be stopped after a preset number of iterations.

The next phase of the Bag of Words consists of utilizing the precalculated types of vectors to create one histogram per word. This histogram ignores the order of appearance of the points, and only stores the number of times each vector has been encountered within the word.

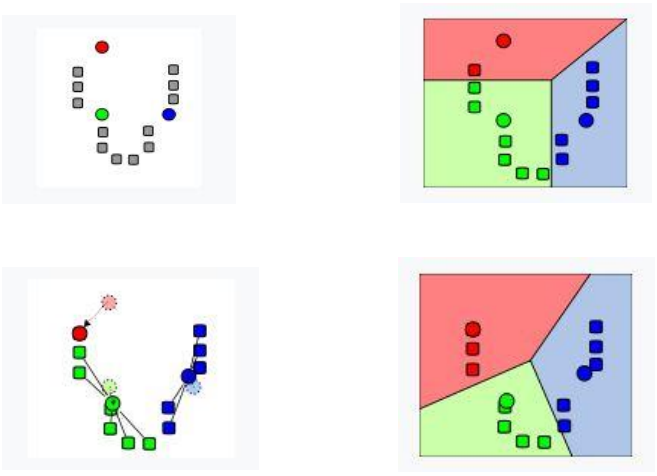

Fig. 10. Example of convergence of the k-means algorithm. In this case, k = 3.

5.4 Creating the recognition system

At this point, we know which parts the recognition system has and how they work in particular. But how do they work together? In this section, you will find the explanation of the recognition system.

In section 5.3, I have explained hos does the core search function works and also the KMeans. Now, I will explain why it is needed. The objective of this algorithm is to be trained outside our application. Once it has been trained with real data, it generates a file containing a preset number of type vectors. The number of vectors (also known as the k of the algorithm) has been established at 150. The file is then moved to our applications and utilized there. This means that our application will not need to execute the KMeans algorithm. It only needs to retrieve the information from the file. I decided to utilize a precalculated file instead of running the algorithm each time the application gets started because it takes too much time, and by doing this, this time is avoidable. There are two main questions: how is the algorithm trained and how it is saved into the file and utilized later.

I have decided to train the algorithm with a single handwritten document. I took this decision because first I wanted to see the results and act based on that. After implementing it and analyzing the results, I decided that this choice would remain intact.
Moreover, to create and save the file and then retrieve the information from it, I have utilized the technique of serializing objects.

After this brief explanation, you will now have all the information needed to understand the following recognition system. The steps are, as follow:

1. Retrieving the data from the file generated by the KMeans.

This first step is executed once the user starts the application.

2. Retrieving the SPenObjectBase objects from the loaded file.
3. Converting those objects into SpenStrokeObjects.
4. Calculating the threshold that I will utilize later to detect words.
5. Calculating the number of total words in the given document.
6. Initializing the histograms for each word. Each of them will start with a frequence of 0.
7. Normalizing the data of the loaded file.

Those steps will be executed when the user loads a handwritten file.

8. Retrieving the SPenObjectBase objects from the user's handwritten note.
9. Converting those objects into SpenStrokeObjects.
10. Normalizing Search data.
11. Creating the feature vectors and updating the corresponding histogram for the given word that will be searched.
12. Normalizing the histograms for the given word that will be searched. This implies changing its frequency to a number within the range of 0 and 1.
13. Creating the feature vectors and updating the corresponding list of histograms for the loaded file.
14. Normalizing the list of histograms for the loaded file.
15. Calculating the similarity between the histogram of the word given and each of the histograms of each of the word detected in the loaded file.
16. Based on the results, changing the ink of the words that meet the requirements of a formula.

Finally, all these steps will be executed when the user has written something and has pressed the button of search. Even though there are a lot of steps, most of them are computed in a few seconds. However, depending on the type of loaded file and the number of objects contained, step 13 may take between 20 seconds and almost two minutes. It should be mentioned that steps 13 and 14 are only calculated after searching a word for the first time. The next search will already have the data calculated. That said, if the loaded file changes, it will have to be calculated again.

5.5 Calculating similarities and creating a formula

In this section, you will find the final stage of the process. At this point, I have calculated all the histograms needed (from the load file and from the given word). In the last step, I have utilized the cosine similarity to

compare histograms. This formula returns a value between -1 and 1, being -1 the opposite, and 1 the same histogram.

First off, I have calculated all the cosine similarity from the histogram that the user typed to each of the words of the loaded handwritten document. Then, I have stored this information sorted from major to minor in two lists, one containing the exact number of the cosinus distance, and the other containing the number in which the word appears in the loaded file. Then, I have applied the next formula:

- All words having a similarity higher to 0.65 are returned.
- The word with the highest similarity is always returned. This word will establish the variable max of the formula
- The words within the next range are returned:

$$(Max - threshold)$$

The threshold has been set to 0.10. This returns the words which similarity is close to the max in the cases where the max is not too high and the word the algorithm is looking for is not clear.

- Finally, there is another threshold involved. If the value of the cosine similarity meets the requirements stated above but is beneath a minimum threshold, it is not returned. This secondary threshold has been set to 0.35. The reason I have introduced this is because in the cases where the max is lower than 0.50, we understand that the word the user was looking for is unclear. Therefore, all that would be below that range, is certainly going to be a mismatch.

So, what happens if the user types something that is not contained in the loaded file? Probably the max is going to be below 0.50. If it is lower than 0.45, we will only return the Max.

In the cases where the same word appears more than once, the algorithm will also detect all of them.

6 Results

In this section, you will find the results obtained from executing the application. The application works most of the times, but not always. There are many factors that affect the correct result of the application such as correctly detecting the words of the loaded file or different handwritten styles.

Figures 11, 12 and 13 are examples of executions with a given word that is contained in the load file. Figure 14 is not contained.

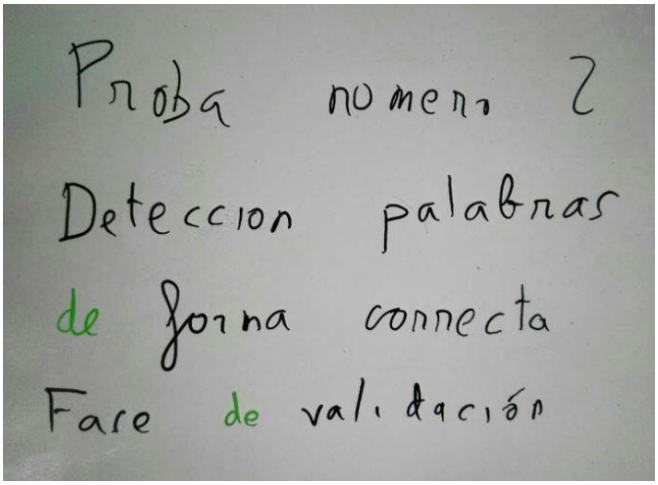

Fig. 11. Example of execution. Searching the given word “de”. Succesful result.

In this case, it detects the word as many times as it appears.

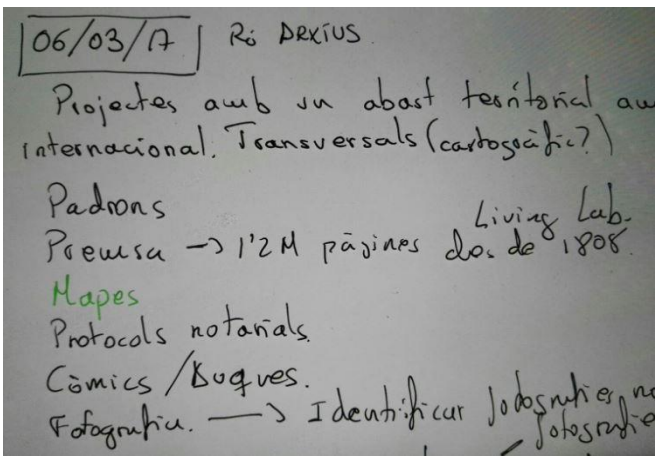

Fig. 12. Example of execution. Searching the given word “Mapes”. Succesful result.

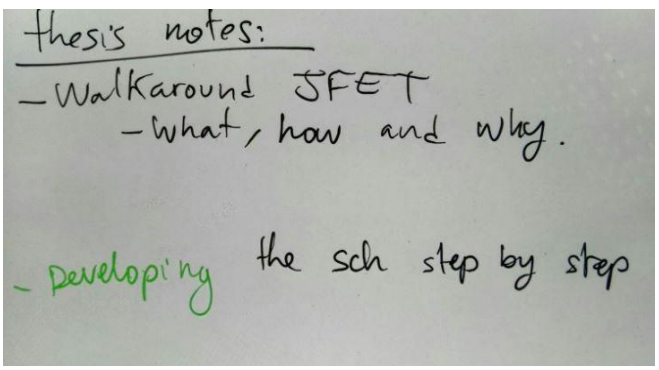

Fig. 13. Example of execution. Searching the given word “Developing”. Succesful result.

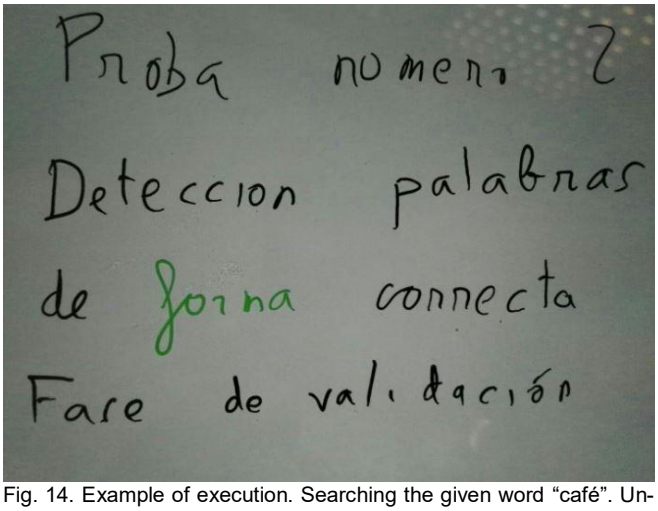

Fig. 14. Example of execution. Searching the given word "café". Unsuccesful result, in this case, the application always returns the word with the highest cosine similarity.

7 Conclusion

Throughout the creation of this project, there have been many changes in the goals and specifications of it. This application searches a word from a loaded file successfully, which was the main purpose of the project, and because of it, this project has been utterly successful. However, there are still some constraints and possible future improvements I would like to talk about.

Firstly, the recognition system would be much more efficient with the complete implementation of the method utilized in the article "Word Spotting in Historical Document Collections with Online-Handwritten Queries" [7]. This project's method utilized a simple method that could be optimized including the complete representation of the attributes contained in the mentioned article. This was not implemented due to the time restrictions this project had. However, this is going to be my main priority to work in the near future.

Secondly, this application only works with one page per document and as of now, it does not let the user change between pages. The explanation for this is simple. Firstly, if the application works with more than one page, the execution times may increase drastically. Remember that they are actually in-between twenty seconds and almost two minutes depending on the loaded file. Secondly, I did not have more time to implement more features such as moving from the pages of the current document. Therefore, this could be seen as another line to work in the future.

Acknowledgment

The author wishes to thank Héctor Miguel Pellicer Grau for supplying him with the hardware needed to develop the application and Ernest Valveny Llobet for his technical assistance.

Bibliografia

[1] Marcus Liwicki, Horst Bunke. "HMM-Based On-Line Recognition of Handwritten Whiteboard Notes". Guy Lorette. Tenth International Workshop on Frontiers in Handwriting Recognition, Oct 2006, La Baule (France), Suvisoft, 2006 <inria-00108307>

[2] S. Jaeger, S. Manke, A. Waibel. "NPen++: An On-Line Handwritting Recognition System". Interactive System Laboratories, University of Karlsruhe, Karlsruhe (Germany).

[3] Nicolas Gramlich, "Andbook", Androir Programming, http://andbook.anddev.org/files/andbook.pdf.

[4] O'Reilly, Ian F. Darwin. "Java Cookbook, Solutions and Examples for java developers", 3rd edition.

[5] Samsung ElectronicsCo. Ltd. "Pen SDK, Programming Guide", version 4.1.2. http://developer.samsung.com/galaxy/pen/guide, 2014.

[6] Samsung SDK API, http://img-developer.samsung.com/onlinedocs/sms/pen/index.html

[7] Christian Wieprecht, Leonard Rothacker, Gernot A. Fink, 2016. "Word Spotting in Historical Document Collections with Online-Handwritten Queries". TU Dortmund University, Dortmund (Germany).