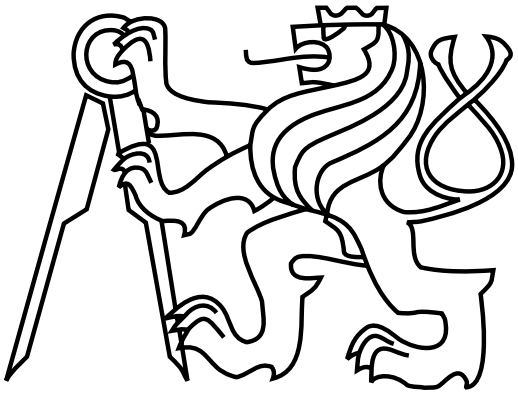
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor Thesis

# Handwriting Beautification

*Ondřej Vančák*

Supervisor: Ing. Šedivý Jan, CSc.
May 2014

**Czech Technical University in Prague**
**Faculty of Electrical Engineering**

**Department of Cybernetics**

# BACHELOR PROJECT ASSIGNMENT

**Student:** Ondřej V a n č á k

**Study programme:** Cybernetics and Robotics

**Specialisation:** Robotics

**Title of Bachelor Project:** Handwriting Beautification

**Guidelines:**

1. Study current methods of handwriting beautification.
2. Implement MATLAB a prototype of one of the State of the Art algorithms.
3. Implement a tablet application for saving handwritten notes.
4. Extend the application of handwriting beautification.
5. Perform a user research (A/B testing) of implemented algorithm.

**Bibliography/Sources:**

[1] Zitnick C. L : Handwriting Beautification Using Token Means. SIGGRAPH, Anaheim, California, USA, 2013.
[2] Schenk J., Lenz, J., and Rigoll G.: "On-Line Recognition of Handwritten Whiteboard Notes: A Novel Approach for Script Line Identification and Normalization," Proc. 11 th Int'l Workshop Frontiers in Handwriting Recognition, Montreal, Quebec, Canada, pp. 540-543, 2008.
[3] Simard P. Y., Steinkraus D., and Agrawala M.: "Ink normalization and beautification," in Proc. 8 th Intl. Conf. on Document Analysis and Recognition (ICDAR 2005), Vol. 2, Los Alamitos, CA: IEEE Computer Society, pp. 1182-1187, 2005.
[4] Vinciarelli A, Luettin J. : "A new normalization technique for cursive handwritten words", Pattern Recognition Letters, Vol. 22, No. 9, pp. 1043-1050, publisher: Elsevier BV,2001.
[5] Zhu X, Jin L. : "Calligraphic Beautification of Handwritten Chinese Characters: A Patternized Approach to Handwritting Transfiguration", International Conference on Frontier on Handwriting Recognition (ICFHR 2008), Anaheim, California, USA, pp. 135-140, 2008.
[6] Brakensiek A, Kosmala A, Rigoll G. : "Comparing Normalization and Adaptation Techniques for On-Line Handwriting Recognition", in Pattern Recognition, Proceeding 16th International Conference on, Quebec, Canada, Vol.3, pp. 73-76, 2002.

**Bachelor Project Supervisor:** Ing. Jan Šedivý, CSc.

**Valid until:** the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
**Head of Department**

prof. Ing. Pavel Ripka, CSc.
**Dean**

Prague, January 10, 2014

**České vysoké učení technické v Praze**
**Fakulta elektrotechnická**

**Katedra kybernetiky**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

**Student:** Ondřej V a n č á k

**Studijní program:** Kybernetika a robotika (bakalářský)

**Obor:** Robotika

**Název tématu:** Zkrášlování rukopisu

**Pokyny pro vypracování:**

1. Nastudujte současné metody pro zkrášlování písma.
2. Implementujte v MATLABu prototyp jednoho ze State of the Art algoritmu pro zkrášlování písma.
3. Implementujte aplikaci pro tablet umožňující vkládání ručně psaných poznámek.
4. Rozšiřte aplikaci o automatické zkrášlování písma.
5. Proveďte uživatelský výzkum (A/B testování) implementovaného algoritmu.

**Seznam odborné literatury:**

[1] Zitnick C. L : Handwriting Beautification Using Token Means. SIGGRAPH, Anaheim, California, USA, 2013.
[2] Schenk J., Lenz, J., and Rigoll G.: "On-Line Recognition of Handwritten Whiteboard Notes: A Novel Approach for Script Line Identification and Normalization," Proc. 11 th Int'l Workshop Frontiers in Handwriting Recognition, Montreal, Quebec, Canada, pp. 540-543, 2008.
[3] Simard P. Y., Steinkraus D., and Agrawala M.: "Ink normalization and beautification," in Proc. 8 th Intl. Conf. on Document Analysis and Recognition (ICDAR 2005), Vol. 2, Los Alamitos, CA: IEEE Computer Society, pp. 1182-1187, 2005.
[4] Vinciarelli A, Luettin J. : "A new normalization technique for cursive handwritten words", Pattern Recognition Letters, Vol. 22, No. 9, pp. 1043-1050, publisher: Elsevier BV,2001.
[5] Zhu X, Jin L. : "Calligraphic Beautification of Handwritten Chinese Characters: A Patternized Approach to Handwritting Transfiguration", International Conference on Frontier on Handwriting Recognition (ICFHR 2008), Anaheim, California, USA, pp. 135-140, 2008.
[6] Brakensiek A, Kosmala A, Rigoll G. : "Comparing Normalization and Adaptation Techniques for On-Line Handwriting Recognition", in Pattern Recognition, Proceeding 16th International Conference on, Quebec, Canada, Vol.3, pp. 73-76, 2002.

**Vedoucí bakalářské práce:** Ing. Jan Šedivý, CSc.

**Platnost zadání:** do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
**vedoucí katedry**

prof. Ing. Pavel Ripka, CSc.
**děkan**

V Praze dne 10. 1. 2014

# Declaration

I hereby declare that I have worked out the presented thesis independently and I have quoted all used sources of information according to the Methodical instructions about ethical principles for writing academic thesis.

In Prague May 23, 2014 .....

# Aknowledgements

I would like to thank Ing. Jan Plešek for his advice and patient guidance of this work. Also I would like to thank Karolina for her psychical support and advice concerning the English language.

# Abstract

The goal of this work is to bring a mobile application that would implement handwriting beautification algorithm. To that account several previously published works on this topic are reviewed and discussed. Selected algorithm is described in depth to provide detailed information of the system principles. The implementation includes MATLAB prototype that has been used for testing. Finally an HTML5 application that beautifies user's handwritten notes has been developed.

# Abstrakt

Cílem této práce je vytvořit mobilní aplikaci, která bude obsahovat algoritmus zkrášlení ručně psaného písma. Za tímto účelem bylo rozebráno a diskutováno několik dosud publikovaných článků s ohledem na jejich možné uplatnění. Vybraný algoritmus byl následně dopodrobna popsán, aby byly osvětleny principy na ktrých celý systém funguje. Implementační část práce zahrnuje prototyp napsaný v MATLABu k otestování algoritmu a finální aplikaci napsanou v HTML5.

# Contents

# List of Figures

# 1 Introduction

Handwritten text still remains one of the most natural ways to write. The benefits of handwriting are indisputable—it is the most comfortable way of taking notes. However, the notes are often not very easy to read, especially when the text is scribed in a hurry. Additionally, in the era of computers the usage of handwritten materials is dropping to digital text, mostly because the computers are usually equipped only with keyboard and are programmed to process textual data more efficiently. With the new tablets and smartphones, however, the touchscreen interface which allows one to write by hand becomes very common. Nevertheless, writing on a touchscreen still is not as comfortable as writing on a piece of paper and the script often is even more difficult to read. There is an open field for software—especially mobile application—that would implement processing methods on the digitalized data and significantly help to increase the handwriting readability.

The usual approach widely used in older devices was based on recognition of the written input. This means the algorithm was interpreting the handwritten symbols and converted them into textual data. Handwriting beautification algorithms, however, usually do not interpret the meaning of the input strokes and only process them to improve their appearance. Today computers, nonetheless, can not imitate the feeling of aesthetics, therefore, the ideal of beautiful script must be defined to them. Definition commonly used in this order is some kind of symmetry although refining based on merging with recognized template is also known.

Even though several beautification systems have been already described in published papers, to my knowledge none of them would be available in the form of an end product software. The goal of this work is to review the state of the art and find a system that could be implemented in a functional application. That means the focus of this work is in realization of the selected algorithm. You can get acquaint with the process of development of the application in the following chapters.

In Chapter 2 the review of several previously published papers on this topic can be found. The algorithm that has been selected as a fundament of the application is explained in detail in Chapter 3. In Chapter 4 the implementation of the algorithm in an application is described. The results of user testing of the application are presented in Chapter 5.

# 2 State of the art

Several articles approached this topic and many of the partial problems are solved and well documented. Nonetheless, most articles deal with the script refining only as a method of data preprocessing. Such preprocessed data is then used only as an input for recognition algorithms. These techniques are described further in Section 2.1.

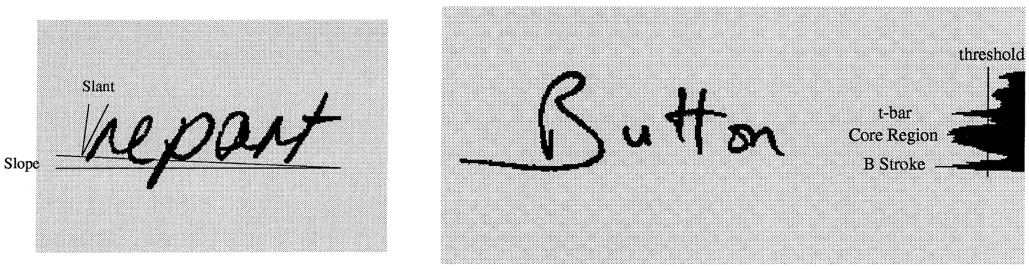
There is also other common approach often used which focuses on rendering of the handwriting and calligraphic simulation. This procedure is well known not only in use in handwriting beautification but also in common note taking and drawing applications. Common approaches are described in Section 2.2.

The approach aimed directly on handwriting beautification is based on refining of the words. In that case user written data are merged with a template and their shape is changed. This approach is rarely found, still, some articles exist and are highly acknowledged in this work. You can find brief explanation of these methods in Section 2.3 and in–depth description of such algorithm used in this work in Chapter 3.

## 2.1 Normalization techniques

Most of the presented techniques are focused on cursive letters only. Mainly in recognition systems it is crucial to normalize the input data in order to achieve concise results. This reflects the fact that most of the handwritten words do not keep the bottom line and the scale varies. Therefore, most of the published articles approach the topic of script reshaping just as a normalization technique used as preprocessing in recognition systems. This task is different form the script beautification for the desired result may not be considered more beautiful by humans. However, the concept is similar and is relevant to the topic.

The article [7] provided a detailed technique for slant and slope removal. The terms slant and slope are well illustrated in Figure 2.1a. This approach works with bitmap renderings of the handwritten text. Described older method for slope removal worked with identified bodies of the letters and applied rotational transformation until the bottoms of the bodies were aligned to horizontal line. The novel presented technique computes density




(a) Slant and slope (b) Density histogram

Figure 2.1: *Illustration of the methods used in article [7]; (a) Illustration of slant and slope on handwritten word. (b) Density histogram computed to identify the slope*

histogram of the written word and applies transformation based on the characteristic peaks generated by the horizontal lines in letters. This method is illustrated in the Figure 2.1b. Also a novel technique for slant normalization that works on similar principle was presented. It was demonstrated that the slant is removed once the maximal of sum of continuous line sections in columns is reached. The previously used technique computed the average slant from the lines connecting the centroids of extreme parts of letters. Similar techniques that also work with bitmap data has been compared in the article [1].

The articles [5] and [2] describe handwriting recognition systems that work with whole word unlike common systems that work with separated letters. Described normalization method works with vectors representing the words. Therefore, further reshaping has less effect on final rendering of the word. Filtering of the input data is applied as to find local maxima an minima in the vertical strokes and reshape them in order to match the writing characteristic lines. The lines used in normalization are illustrated in Figure 2.2; recognized lines within word are shown in Figure 2.3. The normalization method used in system described in the article [4] works on very similar principle with the focus on line normalization of whiteboard notes. All these methods have been proven to increase the classification rate when applied before recognition.

Unlike the other presented methods described above, the article [6] aims not only on preprocessing normalization but also on beautification. Similar method for line fitting is used, however, curvature changes are removed and the script is smoothed in order to produce beautified script and achieve better classification score if used in recognition. Nevertheless, the achieved conclusion is that the line warping is not sufficient for handwriting beautification for the changes are very subtle.

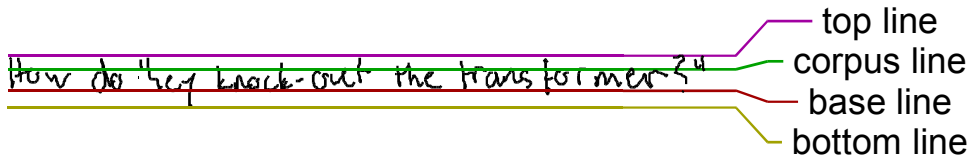



Figure 2.2: *Lines commonly used in line normalization. Corpus line is also often referenced as middle line or half line. Image presented in an article [4]*

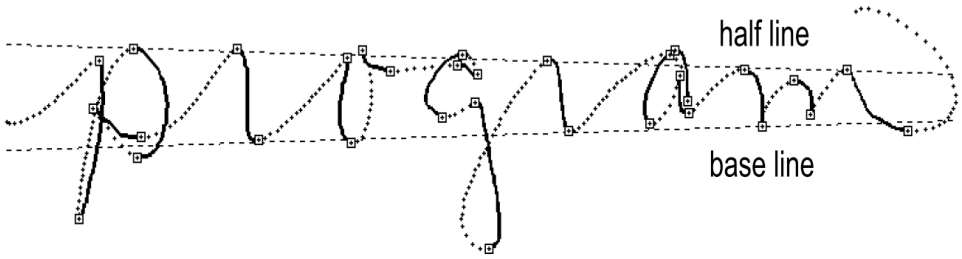


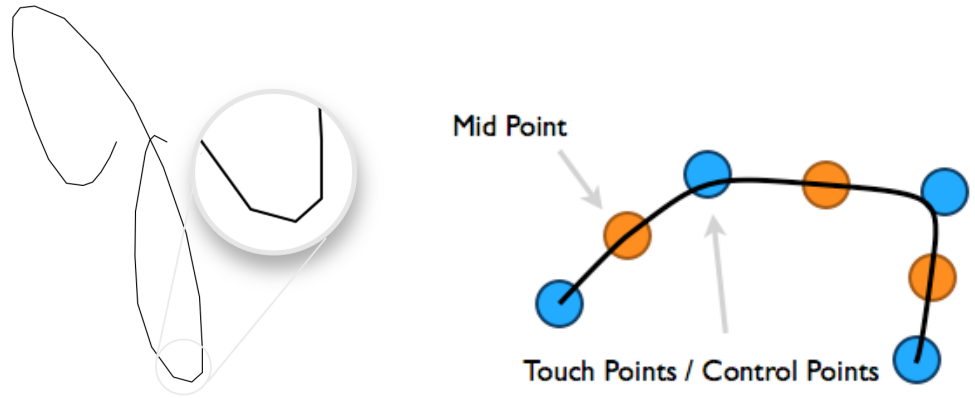

Figure 2.3: *Illustration of recognized local extremes and lines within word. Image taken from an article [5]*

## 2.2 Calligraphic beautification

Especially in Chinese writing the shape of character is not the only thing that matters—the calligraphy can also significantly improve text appearance. The article [8] proposed a method specificly focused on Chinese character beautification. The system is based on both on paintbrush calligraphy simulation and character reshaping. These methods, however, are widely used in many writing or drawing applications and can be applied on any data.

Commonly used calligraphy simulation technique which is also well described in the article [8] is speed based line width regulation. From the collected points the speed of the pen can be estimated easily. The transitions between strokes are smoothed and on the interpolated trajectory between collected points the circles of different diameter are drawn densely. Finally, the anti–aliasing is applied on the rendering in order to smooth the edges of the new trajectory.

The other used technique is based on trajectory interpolation. The data collected from the touchscreen are usually separated points and straight lines are drawn between them like in Figure 2.4a. What the user typically draw, however, is a smooth curve. This shape can be simulated by plotting curves rather than straight lines between the points. A simple approach to this has been presented in [3]. The trick is drawing quadratic lines from the middle




(a) Straight line rendering (b) Quadratic curve interpolation

Figure 2.4: *Illustration of quadratic curve interpolation explained in [3]; (a) The shape of the raw trajectory (b) Quadratic curve interpolation—the original points are used as control points in quadratic bézier curves*

points and using the input points only as control points. This is illustrated in Figure 2.4b. The method proved to be very effective and yet simple, therefore, it was implemented in the application.

## 2.3 Script refining

The approach with the goal to actually beautify the written symbol includes the symbol reshaping. To be able to reshape the symbol the algorithm must have some definition of what the ideal symbol looks like. This could be achieved by improving the symmetry of the symbol—a method used in an article [9] or in the line normalization techniques could be interpreted this way.

In the article [8] a different solution to this problem has been described. The proposed method uses a character recognition to match handwritten Chinese character with its template. A commonly used Chinese font stands as a static library of templates. The recognition is based not only on character shape but also on the order of written strokes which allows several possible sequences. The main advantage of this approach is that it is specialized on Chinese characters only. For Latin letters there is no concise writing style nor there is a usable template for cursive letters. Therefore, a different method of template definition is needed.

The complex solution is provided by the article [9] from Microsoft Research. Their approach works on every possible data be it written text or Chinese characters or mathematical symbols. The main idea of this approach is that when one symbol is written repeatedly, the average shape should be

Figure 2.5: *Illustration of the word "full" being computed as an average of six different instances*

better than individual instances of the symbol—this is demonstrated in Figure 2.5. This approach leads to very robust and complex algorithm that is indifferent to the type of input data and the user's characteristic writing style. For that reason the algorithm has been selected as fundament for the application. The in depth description of the algorithm can be found in Chapter 3; the concrete implementation in the application is presented in Chapter 4.

# 3 | Implemented algorithm

As mentioned in Chapter 2 the method described in an article [9] provided by Microsoft Research has been chosen to be implemented in this work. This method works with so called tokens—the written text is split to constant length stroke groups called tokens and each of these is processed individually. The similar tokens are clustered and if a token is added to the cluster it is reshaped with the mean value of all the tokens in the cluster. This means all the tokens within a cluster look similar but still keep their uniqueness.

As mentioned before the algorithm has been chosen for its robustness—the algorithm works on every input data no matter whether it is written script, Chinese characters or math symbols. This complexity, however, means larger computational load. Therefore, an optimization on the mobile devices is necessary.

The algorithm has been well described in the cited article. In this section it will be described in detail. The flow chart of the algorithm is illustrated in Figure 3.1. The preprocessing methods are explained in Section 3.1, the classification and refining is described in Section 3.2 and the rendering is described in Section 3.3.

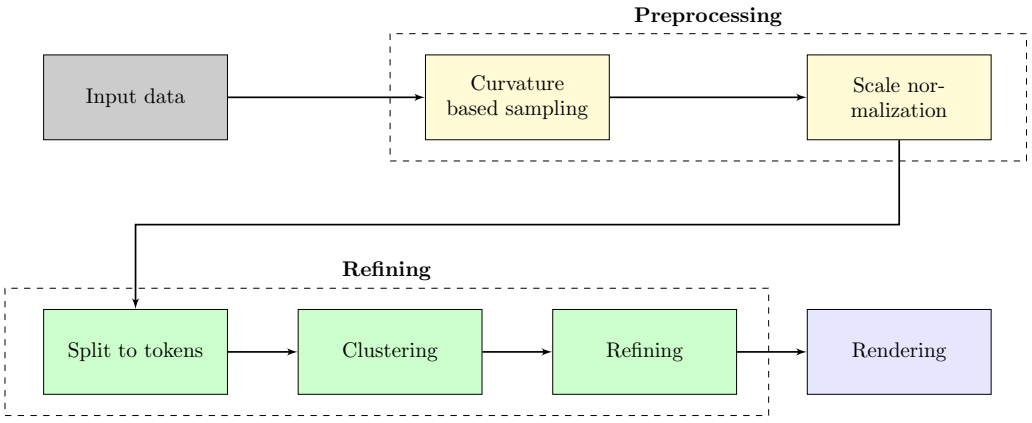



Figure 3.1: *System flow chart*

## 3.1 Data preprocessing

In order to achieve great success with the algorithm, the data preprocessing is crucial. It is important to filter the redundant data and remove the scale variation. The data is stored as difference vectors; the whole set $\Phi = \{\phi_1, \ldots, \phi_n\}$ consists of stroke samples $\phi_i = \{x_i, y_i, p_i\}$ where coordinates $x_i$ and $y_i$ mark the difference between the positions of points on the screen and $p_i$ is the stylus pressure. Since most of today's capacitive screens do not provide pressure information, the value is simply set to $p_i = 0$ when there is no contact with the screen and $p_i = 1$ otherwise. Other often used data references are magnitudes $r_i = \|x_i, y_i\|$ and orientations $\theta_i = \arctan(y_i, x_i)$. Those values could be obtained from $x$ and $y$ values when needed, however, it is faster to compute them immediately when the stroke is captured and save them with the stroke sample.

### 3.1.1 Resampling

The data is usually sampled at uniform distance or in fixed time intervals. The method used in this approach, nevertheless, showed that resampling the data on curvature basis has many advantages over the uniform sampling. With curvature based sampling the sampling rate is decreased on straight line strokes to avoid the redundant data and is increased on curved strokes to preserve the shape information. This leads to a much more concise and better aligned data which are then much easier to process.

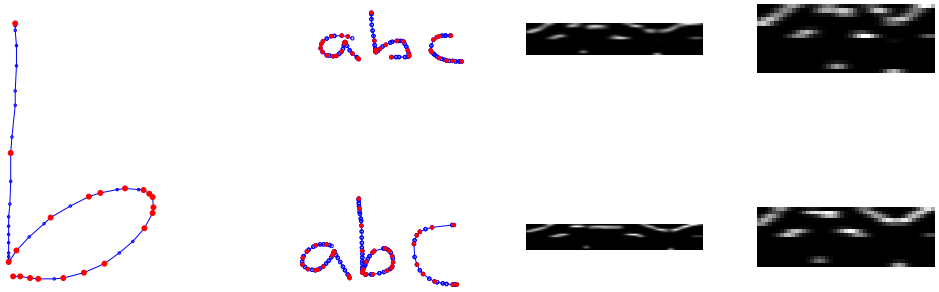
Given the parametrization $\Phi = \{\phi_1 \ldots \phi_n\}$, where every sample consists of coordinates and pressure level $\phi_i = \{x_i, y_i, p_i\}$ the resampled set $\Phi$ is computed. Instead of uniform distance sampling the curvature based samples are found using the Formula 3.1

$$z_i = z_{i-1} + \min(1, \frac{\alpha \Delta_\theta \beta_j}{2\pi}) \tag{3.1}$$

where $z$ is the sample mapping, $\alpha$ is empirically defined constant controlling the sampling density, $\Delta_\theta \in [0, \pi]$ is the absolute orientation change and $\beta_j = \max(0, \min(1, r_j - \sqrt{2}))$ is used to reduce increase of $z$ for small $r_j$. For the pressure information in our case is binary it had to be treated differently. In the article the $p$ value was computed as a weighted average—in our case the the value of $z$ is rounded up with every change of $p$. With the sample mapping $z$ the sample $\phi_i$ is added to sample $\phi^c_j$ using the formula $\phi^c_j = \{x_i + x_j, y_i + y_j, p_j\}$ where $j = \lfloor z_i \rfloor$. If the stroke sample $\phi^c_j$ does not exist yet it is initialized with the value of $\phi_i$. Resampling method is illustrated in the Figure 3.2.

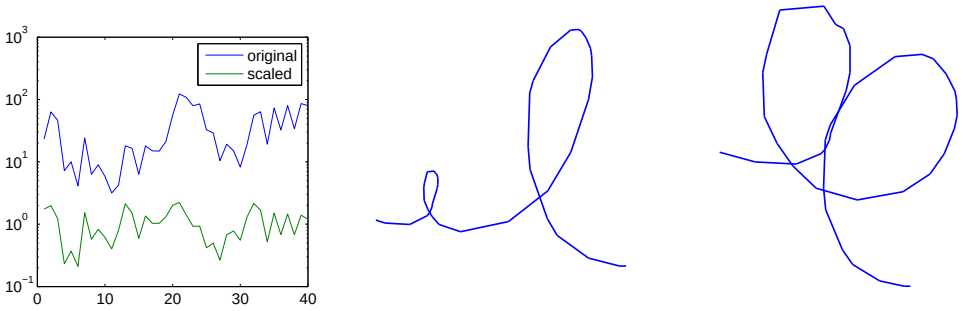
### 3.1.2 Scale normalization

Once the resampled strokes $\Phi^c$ are obtained they are then rescaled in order to remove scale variation between the written input. Gaussian weighted moving



(a) Resampled letter (b) Descriptors of resampled tokens

Figure 3.2: *Illustration of resampling: (a) The sampling ratio is decreasing and increasing depending on the line curvature; (b) The descriptors of two similar tokens are more concise when the data is resampled.*




(a) Scaled magnitudes (b) Input data (c) Scaled data

Figure 3.3: *Scaling of the visible magnitudes: (a) Graph of the magnitudes scale on logarithmic axis; (b) The input data with scale variation; (c) Rescaled data without scale variation*

average is used comparing each sample with the weighted average value of preceding samples. This results in samples with magnitudes $r_i \approx 1$ and only extreme values diverging.

For each visible stroke magnitude $r_i$ the Gaussian weighted moving average value $\eta_i$ is computed using the Formula 3.2. The average value is defined as the sum of the preceding magnitudes weighted by normal distribution function with $\mu = i$.

$$\eta_i = \frac{\sum_{j=i-4}^{i} r_j \, \mathcal{N}(j, i, \sigma)}{\sum_{j=i-4}^{i} \mathcal{N}(j, i, \sigma)} \tag{3.2}$$

The scaled magnitudes are then simply computed as $\hat{r}_i = r_i/\eta_i$ and coordinates likewise $\hat{x}_i = x_i/\eta_i$ and $\hat{y}_i = y_i/\eta_i$. In Figure 3.3 an example of magnitude scaling is shown.

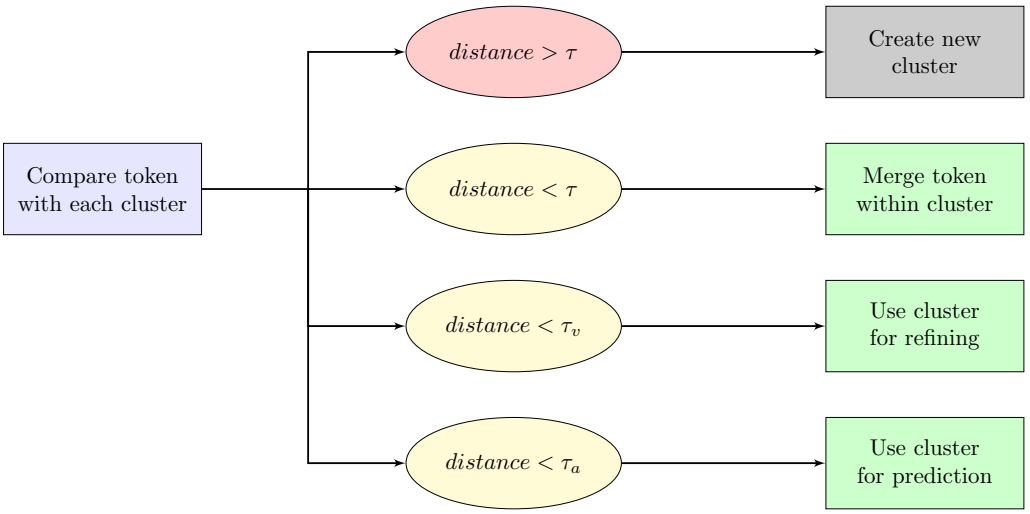



Figure 3.4: *Clustering comparison flow chart*

## 3.2 Token refining

The data consists of large amount of strokes and the spacing between them which also holds important information. The set is split to tokens so that each token consists of $n$ neighboring strokes while each stroke is part of $n$ tokens. Note that the algorithm can run continuously—new data comes as the user writes and is processed immediately.

Each time new token appears its descriptor is computed and it is then clustered. The distance between the token descriptor and the descriptors of cluster means is computed. If the distance is below threshold $\tau$ the token is merged with the cluster mean; otherwise new cluster is created. If the distance is smaller than threshold $\tau_v > \tau$ the cluster is added to a list of potential matches $m_j$ and the token can then be refined if the match is verified. The clusters with distance smaller than $\tau_a$ are added to the list of adjacent clusters and are used for prediction of possible cluster matches of the next token in order to optimize the process. The clustering comparison is illustrated in flow chart in Figure 3.4.

### 3.2.1 Descriptor

The descriptor is used in the metrics of the distance between two tokens in order to decrease the distance between two similar tokens and increase it between the different ones. It provides robustness for small temporal shifts and scale variations and yet it remains characteristic for the input data. In the result stroke magnitudes and orientation changes are involved as well as pen pressure.

For each stroke sample with the magnitude $r$ and orientation $\theta$ the value is computed as a histogram of the orientation multiplied by the magnitude.

(a) Descriptor histogram (b) Descriptor raw (c) Descriptor blurred

Figure 3.5: *Illustration of descriptor: (a) An example of histogram creation; (b) Created descriptor of a token; (c) Final descriptor blurred in order to cover small temporal shifts*

Since the small magnitudes can be as important as the large ones, in this case the logarithmic normalization is applied. Each column of the descriptor has 8 rows for histogram bins at the interval of $\langle 0; 2\pi)$; the value $r$ is split into two bins closest to the value $\theta$ using the ratio of those distances. The histogram creation is illustrated in Figure 3.5a.

Furthermore, a second histogram is computed for the strokes made without the pen being in contact with the screen as those bear significance as well. In order to cover small shifts between two tokens the descriptor is blurred in temporal dimension as shown in Figure 3.5c. The focus in an online refining approach lies on the middle stroke, therefore, the values of the descriptor are weighted by temporally centered Gaussian distribution.

### 3.2.2 Clustering

Clustering is performed dynamically—as the user writes, new stroke samples $\phi$ of the preprocessed data appear. With every new sample a new token can be created and classified. For each stroke sample belongs to $n$ tokens only the first sample of the new token can be refined after the clustering—all $n$ tokens it belongs to have already been clustered.

Crucial role in this procedure is played by the selected value of $n$. Having $n = 31$ showed the best results as each token covered roughly 2–3 neighboring letters. This is very important since the shape of each letter is dependent on adjoining letters but 2–3 letter groups are still generalizable in longer written texts.

In Section 3.2.1 the method of computing a token descriptor has been explained. The $L^2$ distance is used as a metrics of the distance between two descriptors. The clustering procedures dependent on the obtained distance between the token and each cluster is illustrated in Figure 3.4.

### 3.2.3 Merging

Two similar tokens can be merged together to create a new token by averaging their matching points. To find couples of matching points within the tokens a match cost matrix $\beta$ is computed and minimal cost path is found within using a Dijkstra's algorithm to ensure globally optimal path. Thereafter, the corresponding stroke samples are averaged and create new token.

The match cost of two strokes is computed using the Equation 3.3 for $k$th and $l$th member of token stroke sets such that

$$\beta_{k,l} = \Delta_{\hat{r}} + \Delta_{\theta} + \delta_p \tag{3.3}$$

where $\Delta_{\hat{r}}$ is the absolute difference between normalized stroke magnitudes $\hat{r}_k$ and $\hat{r}_l$, $\Delta_{\theta}$ is the absolute difference between $\theta_k$ and $\theta_l$ and $\delta_p = 1$ if both $p_k$ and $p_l$ are either equal to zero or both have nonzero value and $\delta_p = 0$ otherwise.

The optimal path leads from $\beta_{1,1}$ to $\beta_{n,n}$ using only the moves that increase at least one of the coordinates. A move has cost of $\beta_{k,l} + \varepsilon$ where $\varepsilon = 0$ if both coordinates are increased or $\varepsilon = 0.2$ if only one of coordinates is increased. The found path yields pairs of stroke indexes within the tokens and for each pair merged stroke is computed as described in Equation 3.4.

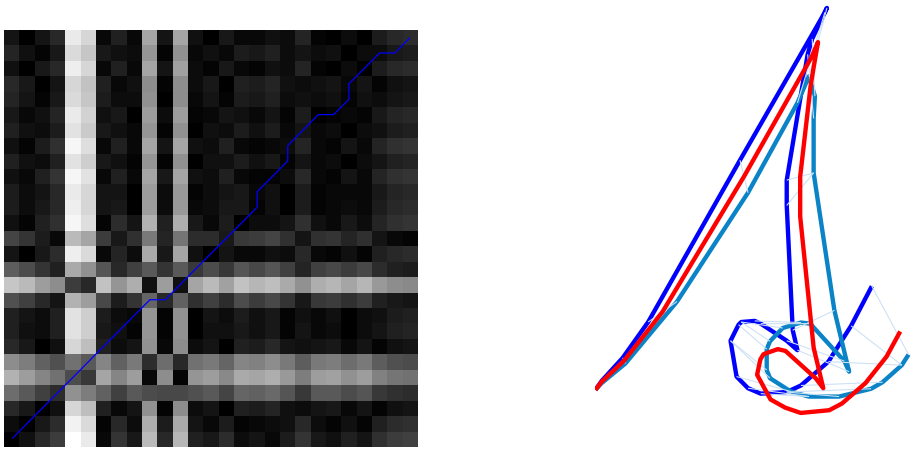
$$\phi_{k,l} = \left\{ \frac{\hat{x}_k + \hat{x}_l}{2}, \frac{\hat{y}_k + \hat{y}_l}{2}, \frac{\hat{r}_k \cdot p_k + \hat{r}_l \cdot p_l}{\hat{r}_k + \hat{r}_l} \right\} \tag{3.4}$$

Since the pressure values are only 0 or 1 in implementation, the value is rounded for the merged strokes with inconsistent visibility. Nevertheless, this happened rarely during the testing.

### 3.2.4 Refining

For each token there is set of clusters which mean tokens are potentially useful for refining. This cluster set consists of all clusters that have the mean within the distance $\tau_v$ larger than $\tau$. Each stroke is refined individually with the cluster matches of all tokens it belongs to. However, the matching clusters for the token have been computed using the token descriptors which can be very similar even if the token renderings would be different. This phenomenon is well illustrated in Figure 3.7. From that reason each cluster match has to be verified with a confidence score. The refines stroke is then computed as an average of corresponding matched strokes weighted by confidence score and the cluster size.

The match confidence score $\lambda_{i,j}$ which is used to verify the match of two tokens $T_i$ and $T_j$ is computed from the blurred token renderings. This ensures the tokens are visually similar. The renderings are spatially centered on the bitmap canvas and then small amount of Gaussian blur is applied on them. The confidence score is then obtained using a normal distribution on the $L^2$ distance of the rendered bitmaps.



(a) Match cost matrix (b) Merged token

Figure 3.6: *Illustration of merged tokens: (a) The match cost matrix $\beta$ with the optimal path marked; (b) The two original tokens (blue) and the product of their merge (red)*

Each stroke of the set belongs to $n$ tokens and each token has a set of candidate cluster mean matches. Therefore, an impact on refining the stroke must be weighted accordingly. The weight $w_{ijk}$ assigned to $i$th stroke in $j$th token for $k$th cluster mean is computed using the Equation 3.5.

$$w_{ijk} = \sum_{j \in [i-(n-1),i]} \sum_{k \in m_j} \lambda_{j,k} \mathcal{N}(i - j, \frac{n}{2}, \sigma) \tag{3.5}$$

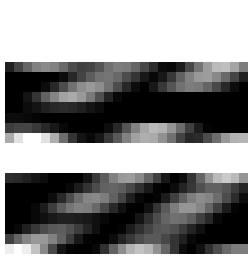
The normal distribution is used to weight the contributions of the clusters and add more importance to those tokens where the stroke is in the middle. The refined stroke is then computed using Equation 3.6.

$$\tilde{x}_i = \frac{\hat{x}_i + \sum_{j \in [i-(n-1),i]} \sum_{k \in m_j} w_{ijk} s_k \hat{x}_{i-j}}{1 + \sum_{j \in [i-(n-1),i]} \sum_{k \in m_j} w_{ijk} s_k} \tag{3.6}$$

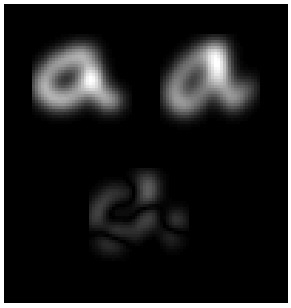
The parameter $s_k$ is used to weight the clusters and is computed as a square root of the cluster size. The value for $\tilde{y}_i$ is of course computed with the same equation.

### 3.2.5 Cluster prediction

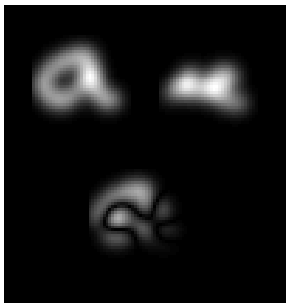
The number of clusters can be really large and comparing each token with every cluster would significantly slow down the whole process. Therefore, an optimization technique is needed. Each cluster has a set of adjacent clusters that are within a threshold $\tau_a$. Since the tokens are overlapping it is expected that the next token would be close to similar set of cluster means.



(a) Descriptors of letters *a* and *ee*



(b) Good match



(c) Bad match

Figure 3.7: *Match confidence illustration—two tokens may have similar looking descriptor and thus be classified within same cluster. Their rendering produced different results, however, and their match confidence $\lambda$ is small.*

By shifting the descriptor of token $T_i$ and comparing it to the set of adjacent cluster descriptors, the set of clusters close to token $T_{i+1}$ is predicted.

Every time a new cluster is created it would be computationally expensive to find adjacent clusters within the whole cluster set. When the cluster set is large only several clusters are selected randomly and if those are close, their adjacent cluster set is compared with the new cluster.

## 3.3 Rendering

Once all the strokes within a stroke set are refined they can be rendered to replace the input data with the beautified version. Each of the visible stroke sets is rendered individually—the non visible strokes are ignored during the rendering process. It is important to align the stroke sets spatially so the new data would be rendered in the same place. Therefore, the stroke set centroid is shifted by the difference to the centroid of matching stroke set within the original data. This procedure is illustrated in Figure 3.8.

Calligraphic simulation techniques mentioned in Section 2.2 can be applied on the aligned rendered data. In this project only quadratic curve interpolation has been used. This technique is dependent on the concrete implementation, therefore, it is described in more detail in Section 4.2.2.

(a) Raw rendering (b) Aligned rendering

Figure 3.8: *Illustration of the rendering alignment; the centroids of the words are highlighted*

# 4 | The application

In Chapter 3 the algorithm fundamental to the app has been described in detail. This chapter reviews to process of implementation of the algorithm. First, the methods have been scripted using the MATLAB environment. The scripting language is easy and works well with large amount of data. Moreover, tuning of the code and parameters is easy due to variety of data visualization techniques. This implementation is more experimental for it does not support data collection and is not conceived as a functional application.

For that a HTML5 application has been written. This app takes an advantage of modern web browsers that are present on many different devices from smartphones and tablets to classic desktop computers. That means the application should be platform independent and usable on any device with a modern web browser. Nevertheless, touch screen devices are of course primarily targeted.

## 4.1 Prototype

The MATLAB has been chosen for the experimental implementation for its easy scripting language and large possibilities of algorithm tuning. However, the scripting language is not fast enough to implement an online working application. Therefore, the algorithm implemented in MATLAB is static and works with the input data provided by a special HTML5 app. Since the computational time of the algorithm increases exponentially for larger data and no special optimizations were applied, the prototype implementation can only handle data consisting of several sentences in the range of minutes. This means the prototype can only be used for algorithm tuning and creation of illustrative figures.

### 4.1.1 The data input

MATLAB supports several common data formats which could be used to load the data input. Nonetheless, this would mean saving to the file and loading it manually every time new data is created. Moreover, this would work only with a special written program on the same computer as the MATLAB is installed. To overcome this problem I have created a simple HTML5

application that allows to capture handwriting on any device with a modern web browser and upload it to custom written Node.js server. From the server the data can be loaded with a single click in the MATLAB integrated web browser as MATLAB provides an interface to launch commands from web page. This also means any function can be launched immediately with the data load to further simplify the process of data import.

### 4.1.2 Parameter tuning

The article that presented the algorithm provided some of the used constant parameters. Nevertheless, they have worked with slightly different input data collected from high resolution pressure sensitive graphic tablet. Therefore, the parameter tuning was necessary to verify the validity of those parameters in this implementation. The MATLAB environment is perfect for the task thanks to large variety of visualization tools and data processing functions.

#### 4.1.2.1 Resampling parameters

The resampling of the data can significantly change the results of the whole system. The process and benefits of curvature based sampling have been described in Section 3.1.1. The sampling density depends on two parameters: $\alpha$ controls the density based on the sample angular differential and $r_\beta$ reduces the sampling of very close samples.

The $\alpha$ roughly corresponds to the number of samples that would represent a circle. The values above 20 have been found sufficient. The proposed value $\alpha = 24$ has been validated by the graph in Figure 4.1a. The input data used for the testing consisted of circles drawn in different scales and for several values of $\alpha$ the average count of samples per circle has been computed.

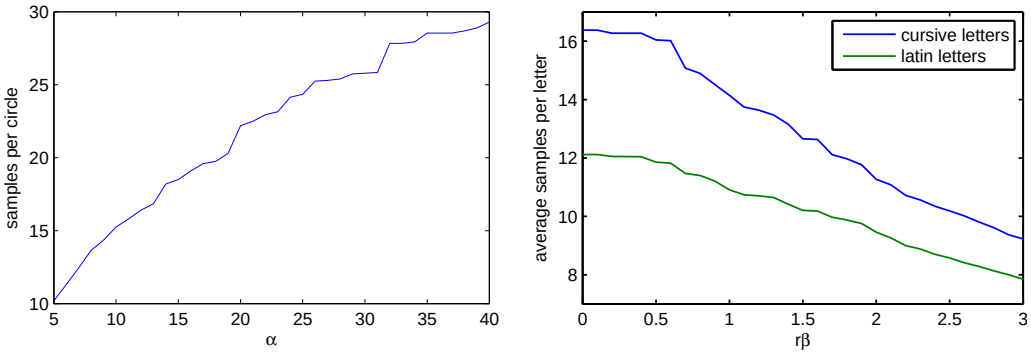
The parameter $r_\beta$ basically controls the smallest distance between two samples. The argument $\beta_j$ used in Formula 3.1 is dependent on parameter $r_\beta$—this is demonstrated in Formula 4.1.

$$\beta_j = \max(0, \min(1, r_j - r_\beta)) \quad \beta_j = \begin{cases} r_j < r_\beta & \beta_j = 0 \\ r_j \leq r_\beta + 1 & \beta_j = r_j \\ r_j > r_\beta + 1 & \beta_j = 1 \end{cases} \tag{4.1}$$

In Figure 4.1b the impact of the parameter $r_\beta$ on the sampling density is illustrated. It shows the the average count of samples per letter is different for the same text written in cursive and Latin scripts. Therefore, values between 1 and 2 are meaningful to keep the suggested rate that 31 samples cover 2–3 letters. Mostly the proposed value $r_\beta = \sqrt{2}$ has been used.

#### 4.1.2.2 Gaussian parameters

In several techniques implemented in the system the Gaussian weightings or blurs are applied. It is hard to determine the impact of their relevant




(a) The impact of $\alpha$ (b) The parameter $r_\beta$

Figure 4.1: *The impact of parameters $\alpha$ and $r_\beta$ on sampling density*

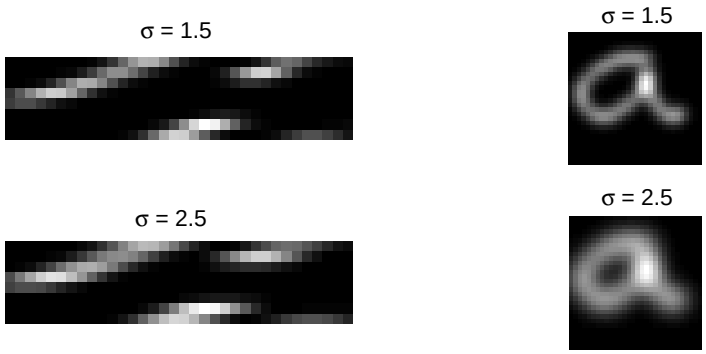



Figure 4.2: *Illustration of the Gaussian parameters: Significant change of the parameter leads to small change in the result*

parameters. Different values have been tried, nevertheless, often the changes were unnoticeable. The proposed values of the parameters have been used for there was no reason to change them. You can find illustration of their impact in Figure 4.2.

#### 4.1.2.3 Clustering parameters

The most important parameter to tune is $\tau$ the maximum distance between the token and a cluster to merge the token within. Values too small mean the effect of beautification is not visible and larger values result in completely deformed shapes that are useless. The testing showed that $\tau$ is very dependent on the nature of input data. It is demonstrated in Figure 4.1b that different scripts have different average count of strokes per letter. This means that the effect of whole beautification process cannot be accurately predicted.

As mentioned above, too large value of $\tau$ can lead to wrong results. Therefore, rather smaller values of $\tau$ were selected to avoid the deformation. The originally proposed range 1.4–1.8 has been decreased to 0.8–1.2, mostly the

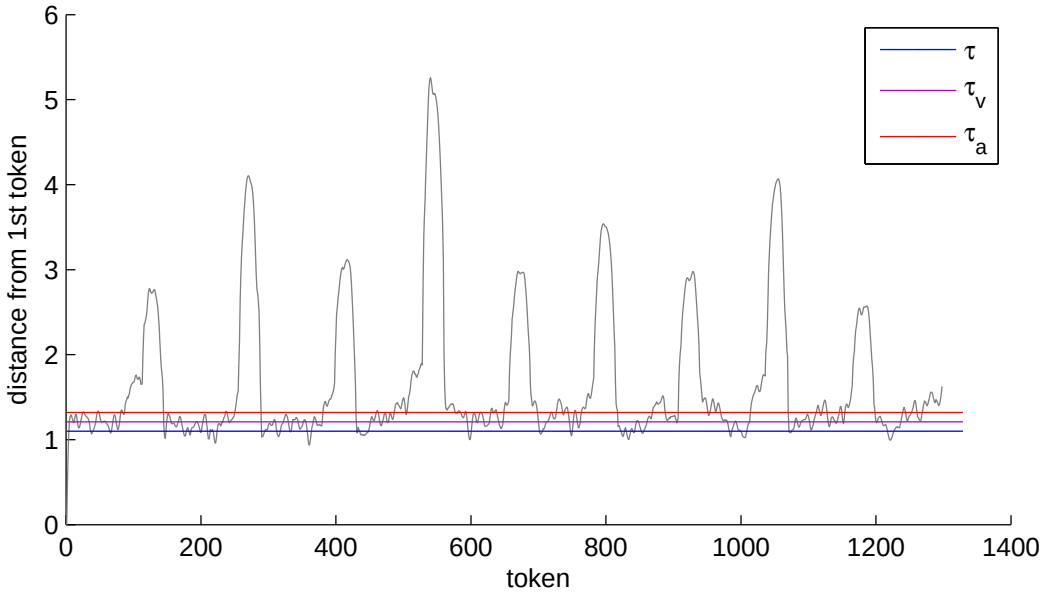



Figure 4.3: *Illustration of the $\tau$ parameter: $L^2$ distance between token descriptors smaller than $\tau$ merges the token with the cluster, smaller than $\tau_v$ are verified and used to refine the stroke, clusters with distance below $\tau_a$ are used to predict close clusters for the next token*

value $\tau = 1.1$ was used. However the proposed ratios for $\tau_v = 1.1\tau$ and $\tau_a = 1.2\tau$ have been kept.

For the parameter tuning mostly the effects of beautification have been rated visually. However, to gain the basic idea of the reasonable value range of $\tau$ the visualization of the distances between token descriptors has been used. The visualization of the distance between the first token descriptor and the descriptors of all other tokens is illustrated in Figure 4.3.

## 4.2 HTML5

To build the application HTML5 has been chosen for several reasons. The main benefit of HTML5 is the ability to run the same application on multiple platforms. That means the application can be used on any device with a modern web browser. Though the javascript performance might not be as good as other languages, the difference should not be problematic for the javascript engines speed has increased dramatically in the last years and in several cases it is comparable to native code. The performance testing is discussed in Section 5.1.

The code has been written using a jQuery framework to interconnect javascript with HTML. The framework has been used mostly within the user interface. The canvas drawing is performed with jCanvas plugin. In order to

achieve better performance, the algorithm itself is written in pure javascript. To save the notes and settings the HTML5 local storage is used.

### 4.2.1 User interface

The application benefits from an HTML5 <canvas> element which allows manual drawing into graphic context. The data is collected by the binded touch/mouse events that provide screen coordinates. Input trajectory is drawn on the canvas immediately to give the user feedback.

The user interface of the application is very simple and allows user to beautify and save notes with different names and load them the next time the application is launched.

### 4.2.2 Rendering

The refined stroke sets have been aligned to their original place using the method described in Section 3.3. In order to smooth the edges the trajectories have been interpolated with quadratic bézier curves using the method described in [3]. The rendering is called once the input data is beautified or when the note is loaded.

# 5 | Results

In Section 4.1.2.3 it was mentioned that the effect of stroke refining is very dependent on the user's script style combined with the tuning of parameters. It has been proven that with parameter $\tau$ set to higher values the results can not be predicted. Therefore, the parameters have been tuned to provide rather subtle but concise results. In Section 5.1 the application is tested in order to prove whether the algorithm is fast enough to run on different devices. To verify whether the subtle changes are sufficient enough to be referred as beautification an A/B testing has been done. In Section 5.2 a user testing results are discussed.

## 5.1 Performance testing

In order to evaluate performance, the algorithm has been tested on several devices. To provide smooth writing experience the application has to be capable of capturing around 50 samples per second to keep the input trajectory information. This leaves the algorithm roughly 20ms of time to refine one sample.

The main bottle neck of the algorithm proved to be the match confidence score computation. To accomplish that the euclidean distance of token renderings has to be found. Even though the modern browsers do support hardware acceleration of 2d graphic context rendering, this task is still computationally expensive because getting the pixel information from the rendered context is costly.

With the cluster prediction only relevant clusters are compared with each token. This means at maximum roughly 30 match confidence score values should be needed to refine the stroke. Therefore, the performance has been tested and evaluated with the average time needed to estimate confidence score. The average has been computed from the overall time taken to compute 10000 scores.

The fastest browser proved to be Internet Explorer 11 which took around 1.5ms per confidence score. This performance is impressive, however, it is still not sufficient for an online working algorithm. The comparison of the performance of different browsers is shown in the chart in Figure 5.1.

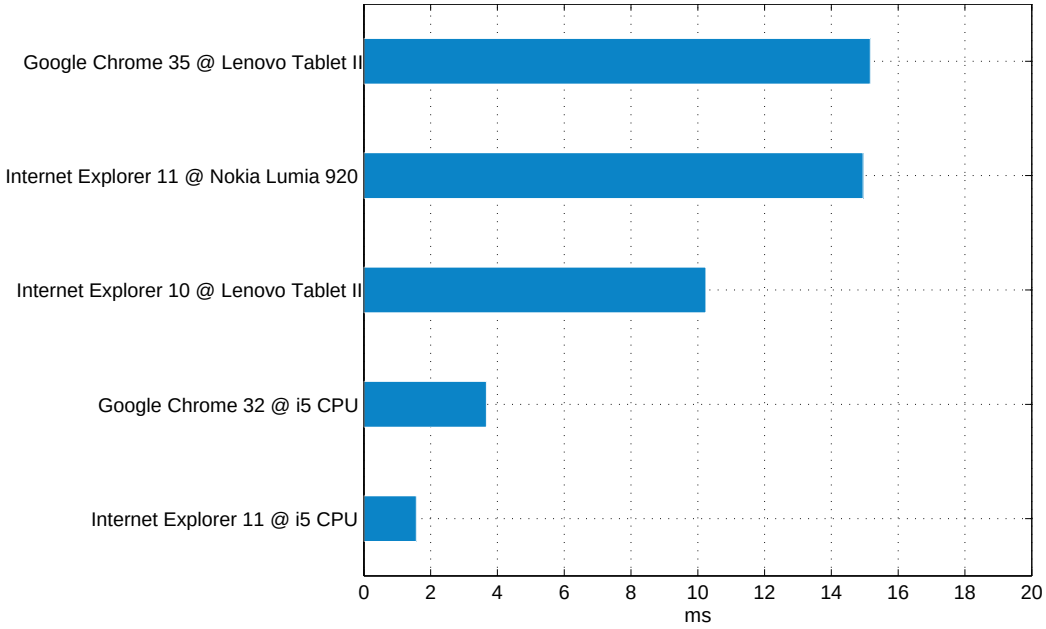



Figure 5.1: *Comparison of match confidence computational performance in different browsers*

## 5.2 User testing

The sense of aesthetics is very individual, therefore, the verification of the results must be done. Several sets of original and beautified pairs of rendering have been produced and the testers have been asked to select which option appears better to them. These tests have been done with vary raw renderings produced by MATLAB prototype and the testers were obliged to select one of the options. The order of the tested samples has been chosen randomly to avoid user decision based on the position.

As mentioned above the algorithm has been set to provide rather subtle changes, therefore, the results of testing are very balanced. This could also be supported with the fact that there has been no option to select neither of the images so the testers decided randomly in the unclear cases. The user testing results are provided by Table 5.1.

| # | original trajectory | beautified trajectory |
|---|---|---|
| 1 | 17 | 25 |
| 2 | 16 | 26 |
| 3 | 32 | 10 |
| 4 | 23 | 19 |
| 5 | 22 | 20 |
| 6 | 12 | 30 |
| 7 | 18 | 24 |
| 8 | 15 | 27 |
| summary | 155 | 181 |

Table 5.1: *User testing results*

Figure 5.2: *Example of tested images, in this case 71% of testers selected the beautified trajectory on the right*

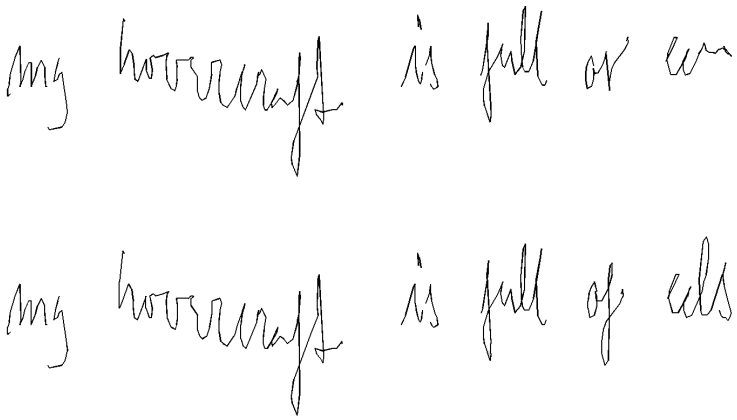


Figure 5.3: *Tested failure case, only 24% testers have chosen the option on top*

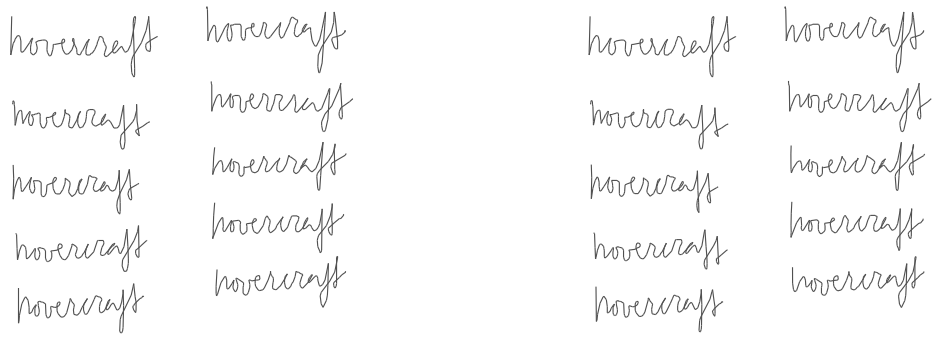


Figure 5.4: *One of the tested images, 64% of the respondents selected the refined image on the right*

# 6 | Conclusion

In this work a process of creating an application with handwriting beautification algorithm has been shown. Several approaches from previously published articles have been discussed and one of them has been selected for the implementation. The selected algorithm has been prototyped and tested within MATLAB interface and coded into HTML5 application. Finally the application has been tested to verify the results.

From the reviewed algorithms the most robust and complex has been chosen in order to bring the best beautification experience. However, the robustness is weighted by the higher computational load which showed to be problematic in an online approach. In future approaches the performance on the mobile devices could be increased by dedicating the calculation on the server.

The algorithm also showed the robustness itself is limited, therefore, the results are dependent on the input data. The parameters have been tuned to provide best results for the possible input data types, nevertheless, in future this issue could be handled by the input data type recognition and dynamic change of the parameters. The parameters tuned to provide concise result for different input styles proved to produce small changes that some of the testers could barely notice.

# CD contents

- /
  - `Thesis.pdf` ..... This work in electronic form
  - `MATLAB` ..... The MATLAB implementation of the algorithm
    - `data` ..... Data used in demo scripts
    - `demo` ..... Demonstration scripts
    - `fig` ..... Scripts used to generate figures
  - `HTML5` ..... The implemented application
    - `index.html`
    - `main.css` ..... Main stylesheet
    - `js` ..... Javascript files
      - `lib` ..... External libraries

# Bibliography


[1] A. Brakensiek, A. Kosmala, and G. Rigoll. Comparing Normalization and Adaptation Techniques for On-Line Handwriting Recognition. In *International Conference on Pattern Recognition*, volume 3, pages 73–76, 2002.

[2] Y. B. Y. L. Cun. Word Normalization for On-Line Handwritten Word Recognition. In *International Conference on Pattern Recognition*, 1994.

[3] J. Harwig. Capture a signature on ios. https://www.altamiracorp.com/blog/employee-posts/capture-a-signature-on-ios, 2013. Accessed: May 2014.

[4] J. Schenk, J. Lenz, and G. Rigoll. On-Line Recognition of Handwritten Whiteboard Notes: A Novel Approach for Script Line Identication And Normalization.

[5] G. Senit, N. NasrabadiS, and R. Sriharit. An on-line cursive word recognition system. In *Computer Vision and Pattern Recognition*, pages 404–410, 1994.

[6] P. Y. Simard, D. Steinkraus, and M. Agrawala. Ink Normalization and Beautification. In *International Conference on Document Analysis and Recognition*, pages 1182–1187, 2005.

[7] A. Vinciarelli and J. Luettin. A new normalization technique for cursive handwritten words. *Pattern Recognition Letters*, 22:1043–1050, 2001.

[8] X. Zhu and L. Jin. Calligraphic Beautification of Handwritten Chinese Characters: A Patternized Approach to Handwriting Transfiguration1.

[9] C. L. Zitnick. Handwriting beautification using token means. *ACM Trans. Graph.*, 32(4):53:1–53:8, July 2013.