Rui Colaço de Almeida

# Human-Computer Interaction: Handwriting Learning

September 2016

Universidade de Coimbra

Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

A Dissertation
for Graduate Study in MSc Program
Master of Science in Electrical and Computer Engineering

# Human-Computer Interaction: Handwriting Learning

Rui Colaço de Almeida

Research Developed Under Supervision of
Prof. Doutor Urbano José Carreira Nunes,
and Doutor Diego Resende Faria

Jury
Prof. Doutor Rui Pedro Duarte Cortesão
Prof. Doutor Gabriel Pereira Pires
Prof. Doutor Urbano José Carreira Nunes

September 2016

*Don't run with the pack.*
*Try something original.*
(James Harris Simons)

# Acknowledgements

Em primeiro lugar, deixo o meu maior agradecimento aos meus pais e irmão. Estou grato pelo seu apoio incondicional e inabalável, em especial agradeço aos meus pais o seu esforço e dedicação diária que permitiram que chegasse até aqui.

I would like to thank my advisors, Dr. Diego Faria and Professor Urbano Nunes, for their never ending availability to provide support during this work.

I thank to my laboratory colleagues, in special to my friend and closest working partner José Medeiros, for making this journey very fun. To all my friends that, just like me, made the final dissertation this semester, in special my great friend João Toscano, I am grateful to have shared this special academic moment with you.

I would also like to thank ISR for hosting me, providing the necessary resources, conditions and personnel that allowed me to accomplish all the goals I worked for. This work has been supported by Fundação para a Ciência e Tecnologia (FCT), COMPETE and QREN programs, under the project "AMS-HMI12 - Assisted Mobility Supported by shared control and advanced Human Machine Interfaces" with reference RECI/EEI-AUT/0181/2012.

To all my friends, to those that are with me since forever and to those that have appeared throughout my academic life, a great warm *Thank You!*

# Abstract

Handwriting skills have a major impact in the self-esteem of young children. Furthermore, good handwriting skills is considered a fundamental step to future academic success [3, 17]. Children with writing disorders had propensity to inferior verbal IQ, weakened mathematics abilities and higher attention difficulties [16]. Considering that writing disorders affect approximately 10 to 30% [8] of children it, is easy to understand that **efficient** handwriting therapy is of paramount importance.

'Conventional therapy' consisting of many training sessions, and 'Learning by Teaching' in which the student teaches a colleague in training sessions, are two proposed successful therapy techniques [7, 15]. In this work we involve both techniques in the effort to create a software framework that will serve as foundation for a future humanoid learning/teaching partner. This partner aims at being an extra tool for the handwriting therapist, not a substitute. Using the framework it will be possible to operate any of the therapy paradigms.

The implemented system uses a dataset [11] to produce a **signature** for each character writing variation. These signatures, called **master signatures**, are compared with the user input to classify it. Having the user input classified it is possible to morph it towards a more correct character (for conventional teaching) or use it to improve a less correct character (for learning by teaching).

The performance of the proposed system was assessed and validated. It was possible to prove that shape information is fully preserved and used in the classification, not only enabling to divide characters of different variations, but also allowing to divide characters of the same variation but with different drawing path. This capability allows the master characters to be displayed (not instantaneously but) as being written by a human intervener, enabling a very straightforward future integration with a humanoid robot. The results also proved the ability of the framework to receive a user input, interpret it and produced a didactic output in line with the therapy technique being used. The framework developed fully meets all the objectives defined.

**Keywords:** Autonomous Handwriting Learning; Humanoid Robot; Learning By Teaching; Automatic Shape Classification; Automatic Shape Recognition.

# Resumo

As habilidades de escrita de uma criança em idade escolar têm um grande impacto na sua auto-estima. Além disso, boas habilidades de escrita são consideradas um requisito fundamental para o seu futuro sucesso acadêmico [3, 17]. Crianças com dificuldade na escrita têm propensão a sofrer de um reduzido QI verbal, aptidão matemática enfraquecida e déficit de atenção [16]. Considerando que as dificuldades de escrita afetam aproximadamente 10 a 30 % [8] das crianças, é fácil entender que uma terapia **eficiente** é de suma importância.

A 'terapia convencional' que consiste em muitas sessões de treinamento, e a terapia *'Learning by Teaching'*, no qual o aluno ensina um colega em sessões de treino, são duas técnicas de terapia propostas com sucesso comprovado [7, 15]. Neste trabalho ambas as técnicas são incluídas num esforço para criar um *software framework* que servirá como base para um futuro parceiro humanóide de aprendizagem/ensino caligráfico. Este parceiro pretende ser uma ferramenta extra para o terapeuta de escrita, não um substituto. Usando o *framework* que vai ser possível operar com qualquer uma das técnicas de terapia.

O sistema implementado utiliza um *dataset* [11] para produzir uma **assinatura** para cada variação escrita de uma letra. Estas assinaturas, designadas **assinaturas mestre**, são comparados com o *input* do utilizador para o classificar. Uma vez classificado, é possível aproximá-lo de um caractere mais correto (para o ensino convencional) ou usá-lo para melhorar um caractere menos correto (para a *learning by teaching*).

O desempenho do sistema proposto foi avaliado e validado com testes. Foi possível provar que a todas a informação sobre o formato dos caracteres é completamente preservada e utilizada na classificação, não só permitindo dividir caracteres de diferentes variações, mas também permitindo dividir caracteres da mesma variação, mas desenhados usando caminhos diferentes. Esta capacidade permite que as letras sejam escritas (não instantaneamente, mas) como se estivessem a ser desenhadas por um interveniente humano, possibilitando no futuro uma integração muito simples de um robô humanoide no sistema.

Os resultados também demonstraram a capacidade do framework para receber o input do utilizador, interpreta-lo e produzir um output didáctico de acordo com a técnica de terapia a ser utilizada. O *framework* desenvolvido cumpre totalmente todos os objectivos definidos.

**Palavras-chave:** Aprendizagem Caligráfica Autónoma; Robot Humanóide; Learning By Teaching; Classificação Automática de Padrões; Reconhecimento Automático de Padrões.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **aaad** | Average Absolute Amplitude Difference |
| **APC** | Adapted Polar Coordinates |
| **DMP** | Dynamic Movement Primitives |
| **DTW** | Dynamic Time Warping |
| **HMM** | Hidden Markov Model |
| **IOC** | Inverse Optimal Control |
| **MCL** | Markov Clustering Algorithm |
| **PCA** | Principal Component Analysis |
| **SAM** | Signature Adjacency Matrix |
| **SDM** | Signature Difference Matrix |

# Chapter 1

# Introduction

From young age children develop a perception of their overall self-worth as well as domain-specific assessment of their competence, being scholastic competence one of this domain-specific assessments - chapter 5 [1]. Since between 31 to 60% of a children's school day is spent in handwriting and other fine motor skills [12], handwriting skills have a major impact in the self-worth evaluation in these ages. Furthermore, good handwriting skills development is far from being only important in self-esteem building, being considered a fundamental step to future academic success [17, 3]. Children with writing disorders had propensity to inferior verbal IQ, weakened mathematics abilities and higher attention difficulties [16].

## 1.1 Motivation

Considering that writing disorders affect approximately 10 to 30% [8] of children it, is easy to understand that **efficient** handwriting therapy is of paramount importance. Common successful interventions consist of a great number of regular training sessions where the skill is physically practised [7]. This might lead to a series of logistic problems if not enough specialized personal is available to fully accompany all children at all times, might also lead to a demoralized child that feels obligated to do something that he does not feel he is good at. Although the conventional therapy technique is always necessary, another technique called 'Learning by Teaching' might mitigate some of the referred problems by keeping the student engaged in the activity while his attention is transferred to teaching a colleague, this has been shown to produce meta-cognitive, motivational and educational benefits [15].

In this work we involve both handwriting therapy techniques (conventional and learning by teaching) in the effort to create a software framework that will serve as foundation for a future humanoid learning/teaching partner. This partner aims at being an extra tool for the handwriting therapist, not a substitute. Using the framework it should be possible to operate any of the therapy paradigms interacting with the student via a digital tablet.

## 1.2 Objectives

The primary goal of this dissertation is to develop a software framework foundation for a robotic handwriting learning partner for children with handwriting difficulties. The handwriting learning support system will consist of a digital tablet connected to remote software capable of interpreting the user's input, simulating handwriting and improving its calligraphy based on suggestions made by the user (learning by teaching), as well as suggesting improvements for the user's input (conventional teaching). This system in future iterations will be augmented with a friendly looking humanoid robot, therefore the characters should be displayed on the digital tablet (not instantaneously but) as being written by a human intervener, to allow easy integration with the humanoid in the future.

## 1.3 Implementations and key contributions

The focus of this dissertation was the development of a complete software framework (and mathematical tools) of an autonomous handwriting learning support system. All the framework components created, described in figure 1.1, were tested using publicly available datasets of characters [4, 11]. The results show a good performance, validating the system.

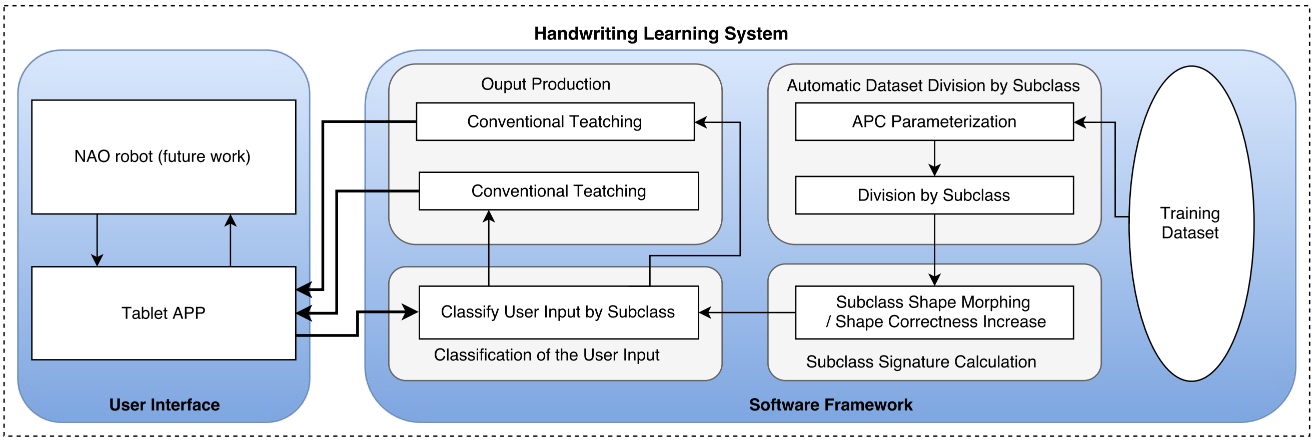

Handwriting Learning System
NAO robot (future work)
Tablet APP
User Interface
Ouput Production
Conventional Teatching
Conventional Teatching
Classify User Input by Subclass
Classification of the User Input
Automatic Dataset Division by Subclass
APC Parameterization
Division by Subclass
Subclass Shape Morphing / Shape Correctness Increase
Subclass Signature Calculation
Training Dataset
Software Framework

Figure 1.1: Implementations and key contributions

The implementations and key contributions of the presented work are the following.

**Automatic Dataset Division by Subclass (Section 3.2):**

- Creating a parameterization technique for 2D shapes;
- Developing a methodology to automatically divide a shape dataset into subclasses - taking into account shape morphology and drawing path;

**Subclass Signature Calculation (Section 3.3):**

- Developing a technique to morph two or more characters into one, allowing to choose how much each character contributes for the morphed character;
- Creating a process to generate high calligraphic correctness characters;

**Classification of the User Input (Section 3.4):**

- Developing a technique to classify the user input by shape morphology;
- Creating a method to evaluate a character calligraphic correctness;

**Output Production (Section 3.5):**

- Developing a process to generate low calligraphic correctness characters;

# Chapter 2

# Background and State of the Art

## 2.1 Background

### 2.1.1 Modified Dynamic Time Warping

The conventional DTW algorithm is used to measure the similarity between two time series, by assuming that one is a non-linear time-stretched version of the other and that their actual values are on the same scale. The algorithm tries to find a non-linear way of stretching the sequences so that an optimal match is achieved. Considering two temporal sequences $\mathbf{a} = a_1, a_2, \ldots, a_n$ and $\mathbf{b} = b_1, b_2, \ldots, b_m$ a distance $d(i, j)$ is calculated for every sample pair $(a_i, b_j)$, $i = 1, \ldots, n$ and $j = 1, \ldots, m$, usually Euclidian distance is used, i.e. $d(i, j) = |a_i - b_j|$. The distance values are stored in a $n \times m$ distance matrix $D$, in which $D_{i,j} = d(i, j)$. In order to create a mapping between the two signals a path from $D_{1,1}$ to $D_{n,m}$ needs to be defined. Since each matrix entry represents the distance between the two points of the sequences, the sum of all entries along the path defines the distance between the new mapped signals. The objective is to find the path that minimizes this distance. The path choices usually obey several conditions, of which monotony (indexes $i$ and $j$ must stay the same or increase but never decrease) and continuity (indexes $i$ and $j$ can never increase more than one step at a time) should be noted. An approach to find the best path is to construct an accumulated distance matrix $AD$, in which the entry $AD_{i,j}$ contains the minimum distance to reach $D_{i,j}$ from $D_{1,1}$. This is done using the following formula $AD_{i,j} = min(AD_{i-1,j}; AD_{i,j-1}; AD_{i-1,j-1}) + D_{i,j}$. Backtracking by finding the next smallest neighbouring value from $AD_{n,m}$ to $AD_{1,1}$, allows to find the optimal warp path.

For two sequences $\mathbf{a} = \{1,1,1,2,1,1,2,3,2,0,1\}$ and $\mathbf{b} = \{1,2,1,2,3,2,1\}$ figure 2.1 shows $D$

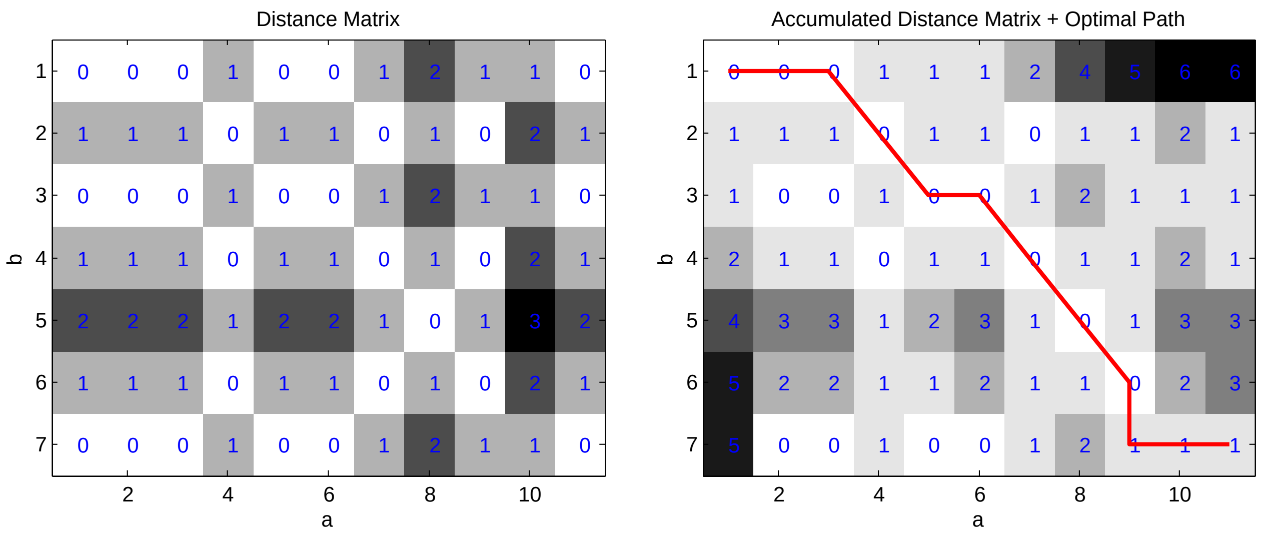


Figure 2.1: Distance matrix, Accumulated distance matrix and Optimal path (in red) for a DTW of signal *a* and *b*.

and *AD* matrices and the optimal path obtained using the algorithm. For this example the resulting $a_{path} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11\}$ and $b_{path} = \{1, 1, 1, 2, 3, 3, 4, 5, 6, 7, 7, 7\}$ are obtained from the *AD* matrix indexes and store the correspondence between the elements of the sequences, figure 2.2 shows this correspondence visually.

### 2.1.2 Adjacency Matrix

In graph theory, a finite graph is a set of nodes connected together by edges (figure 2.3). If the edges are weighted they quantify the relationship between the nodes they connect. When edges weight is the same in both directions, the graph is said to be undirected. A graph can be represented by a square matrix designated adjacency matrix, if the graph is undirected the matrix is symmetric. The matrix entry $(i, j)$ quantifies the weight of the edge that connects node $i$ with node $j$ as shown in figure 2.3. For an adjacency matrix the higher the value of entry $(i, j)$ the most probable is that a random process would jump from node $i$ to node $j$ or vice-versa, therefore the higher the values the more related two nodes are.

### 2.1.3 Markov Cluster Algorithm

For a graph, there are usually groups of nodes with relatively high weight edges between all elements of the group, this groups are called clusters. The opposite is also true, the edges outside of clusters

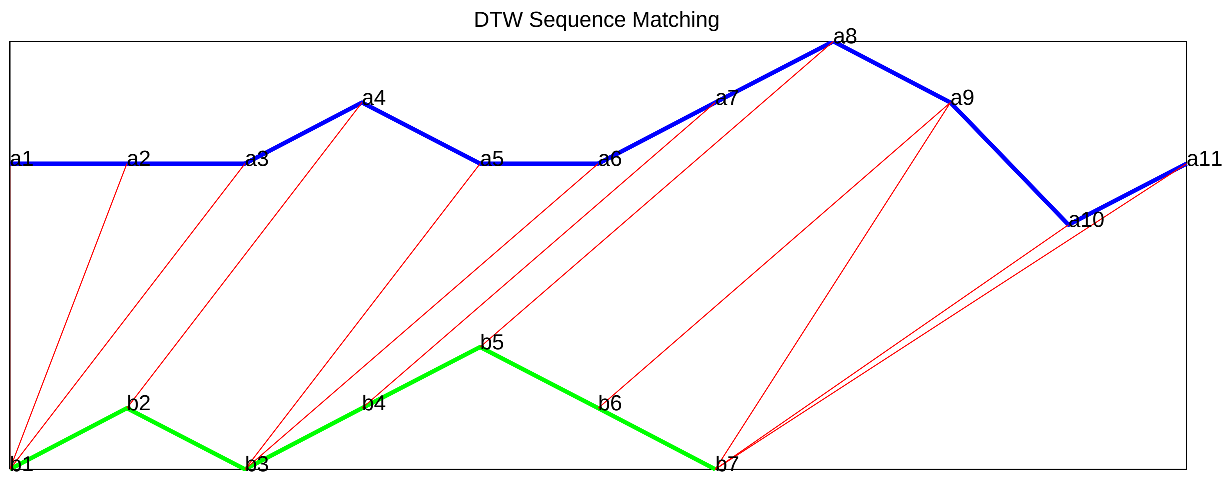
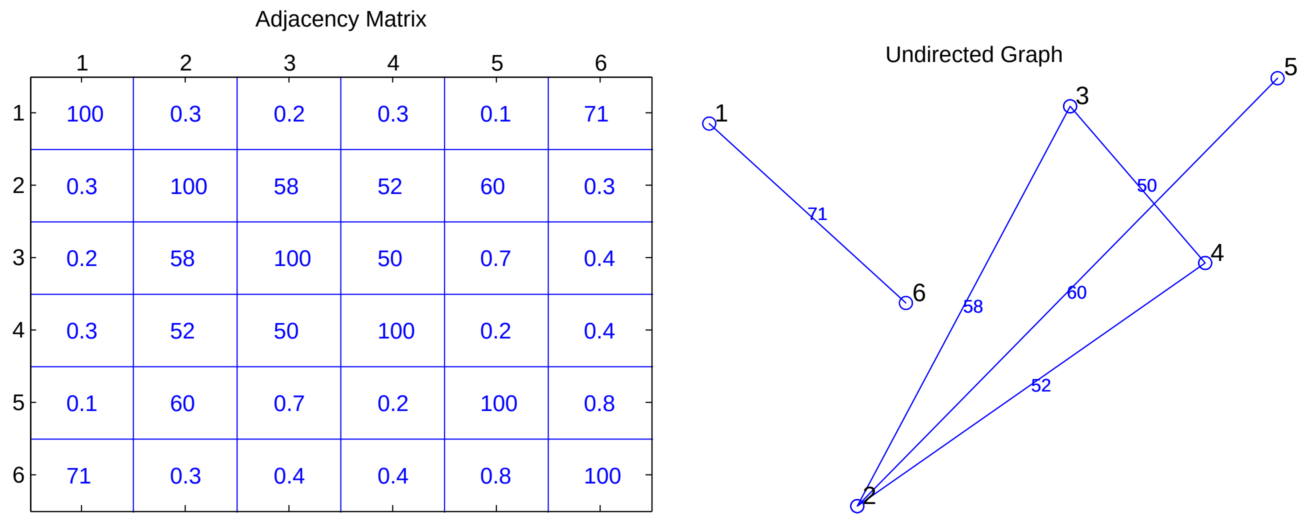


Figure 2.2: Optimal match between the two sequences, being *a* the blue sequence and *b* the green.



|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 100 | 0.3 | 0.2 | 0.3 | 0.1 | 71 |
| 2 | 0.3 | 100 | 58 | 52 | 60 | 0.3 |
| 3 | 0.2 | 58 | 100 | 50 | 0.7 | 0.4 |
| 4 | 0.3 | 52 | 50 | 100 | 0.2 | 0.4 |
| 5 | 0.1 | 60 | 0.7 | 0.2 | 100 | 0.8 |
| 6 | 71 | 0.3 | 0.4 | 0.4 | 0.8 | 100 |

Figure 2.3: Symmetric adjacency matrix and correspondent undirected graph. Each row or column of the table correspond to a node in the graph. Edges with insignificant edge weight ($\approx 0$) were ignored when ploting the graph.

have relatively low weights [1]. As a consequence of this, executing a 'random walk' between nodes, one is more likely to stay within a cluster than going out, this holds clues to where most flow gathers helping to estimate the said clusters. Normalizing the adjacency matrix of a graph by column results in a probability matrix (stochastic matrix) associated with the transitions between nodes. When the weight of the graph's edges are represented by this probabilities, the graph is called a 'Markov Chain'. The 'Markov Cluster Algorithm', formally defined by Dongen [19], is based upon this definitions and properties. Citing the author, "[the algorithm] does not contain high-level procedural rules for the assembling, splitting, or joining of cluster" it is therefore elegant and straightforward, working in three fundamental iterative steps: Normalization; Expansion and Inflation. Before applying this steps, an optional preparation of the adjacency matrix consisting of adding a fixed value $a$ in all entries of its diagonal, called self-loop weight, allows to increase granularity by reinforcing the effect of each node belonging to a cluster composed of itself (in the extreme case $a \to \infty$ the final cluster arrangement for $n$ nodes consists of $n$ clusters with one node each).

1. **Normalization** is the process of normalizing a matrix by column (dividing each element by the sum of its column), always resulting in a left stochastic matrix. This evidences the probabilistic distribution of flow converging/diverging from each node.
2. **Expansion** is the process of computing the $e^{th}$ power of the matrix resulting from the previous operation. The value $e$ is called the expansion factor. This operation allows the flow to connect to other regions of the graph.
3. **Inflation** is the process of computing a element wise $r^{th}$ power of the matrix from the previous operation. The value $r$ is called the inflation factor. This operation 'strengthens the strong edges' and further 'weakens the weak edges' in each node. This concept might not be intuitive, but can be easily understood by a simple example. Considering the vector [0.75; 0.25] representing a column of a $2 \times 2$ stochastic matrix. The first element has triple the magnitude of the second, therefore the first represents a stronger edge. After quadratically exponentiating and normalizing, the resulting vector is [0.9; 0.1] it is easy to verify the first element now has nine times the magnitude of the second, i.e. 'the strong edge was strengthened and the weak further weakened'.

The existence of both operations (expansion and inflation) is fundamental to obtain convergence towards the result, however the "inflation parameter $r$ is the main factor influencing granularity of the

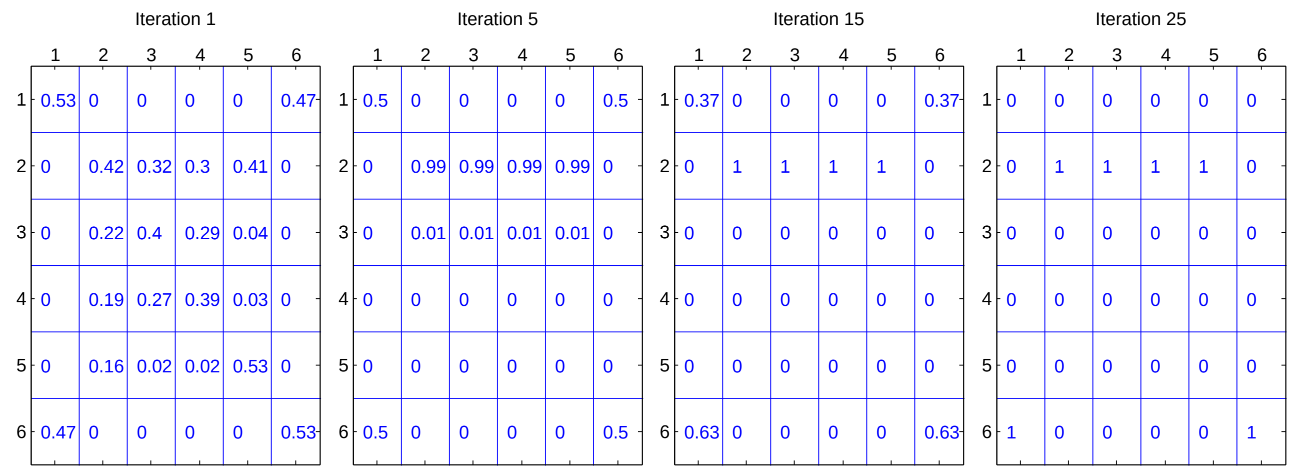
[1] The definition of 'high' and 'low' weight is not objective and introduces us to a common problem described in literature when defining clusters, which are usually defined as "natural groups within a class of entities", chapter 1 [19]. The word 'natural' isn't much more objective itself. Later in this section one will understand that the admissible 'separation' between clusters, i.e. granularity, is variable and can be finely tuned using the input parameters of this algorithm.

resulting clusters” (chapter 10 [19]). By increasing $r$ the final clustering granularity is increased and vice-versa. Algorithm 1 defines formally the MCL version used in the work.

**Algorithm 1:** Markov Cluster Algorithm

```
input  : An adjacency matrix $ADJm \in \mathbb{R}^{n\times n}$
         An self-loop weight $a$
         An inflation factor $r$
         An expansion factor $e$
         A maximum iteration parameter $maxIt$, that defines when to return if convergence is not
         reached
$ICDm = -1$;
$ADJm = ADJm + a.I$ ;                         // $I \in \mathbb{R}^{n\times n}$ is an identity matrix
$ADJm = norm(ADJm, 2)$ ;       // $norm(X,2)$ performs column-wise normalization of
matrix $X$
for  i = 1 to maxIt do
    $PrevADJm = ADJm$;
    $ADJm = (ADJm)^e$;
    for every element $ADJm_{i,j}$ do
        $ADJm_{i,j} = (ADJm_{i,j})^r$;
    end
    if $ADJm == PrevADJm$ then
        $ICDm = ADJm$;
        break;
    end
end
output: An incidence matrix $ICDm$, in which and entry $ICDM_{i,j} > 0$ indicates $i$ and $j$ belong to
         the same cluster
```

As an example, the MCL algorithm with parameters $a = 0$, $e = 2$ and $r = 2$ was applied to the matrix from figure 2.3 resulting in the clustering $\{1,6\}$ and $\{2,3,4,5\}$, for the graph showed in the same figure, as expected. Several iterations of the process are show in figure 2.4.



Iteration 1

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.53 | 0 | 0 | 0 | 0 | 0.47 |
| 2 | 0 | 0.42 | 0.32 | 0.3 | 0.41 | 0 |
| 3 | 0 | 0.22 | 0.4 | 0.29 | 0.04 | 0 |
| 4 | 0 | 0.19 | 0.27 | 0.39 | 0.03 | 0 |
| 5 | 0 | 0.16 | 0.02 | 0.02 | 0.53 | 0 |
| 6 | 0.47 | 0 | 0 | 0 | 0 | 0.53 |

Iteration 5

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.5 | 0 | 0 | 0 | 0 | 0.5 |
| 2 | 0 | 0.99 | 0.99 | 0.99 | 0.99 | 0 |
| 3 | 0 | 0.01 | 0.01 | 0.01 | 0.01 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0.5 | 0 | 0 | 0 | 0 | 0.5 |

Iteration 15

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.37 | 0 | 0 | 0 | 0 | 0.37 |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0.63 | 0 | 0 | 0 | 0 | 0.63 |

Iteration 25

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 0 | 0 | 0 | 0 | 1 |

Figure 2.4: Several iterations of the MCL algorithm applied to the matrix/graph in figure 2.3. Note that in the final iteration, for each cluster a central node is defined, that is the node of the row that stores the incidence values (for this example, nodes 2 and 6).

### 2.1.4 Chord-length Distribution

Techniques from several areas of science hold this designation as result of being based in some form of chord-length distribution characterizing a certain object, shape or substance. A reference to a technique similar to the used in this work, being however used with different purpose, was found in Yang [20] and designated as ‘proportional chord-length principle’.

Using the process described in section 2.1.1, it is possible to execute automatic continuous feature matching of two shapes. The chord-length distribution allows to compare the similarity of two shapes whose features might match despite being ‘visually dissimilar’, figure 2.5 exemplifies the technique for two shapes with identical angle-wise features but with some degree of visual dissimilarity. For ease of understanding only four features were used for this chord-length distribution, but all points of a shape can be used as a feature for this technique (i.e. continuous feature matching). The main conclusion to be drawn from the example, is that if both shapes were visually similar the magenta dashed line should be nearly horizontal with all values close to zero. The deviation from this characteristics can be quantified mathematically and therefore be used as a metric for visual similarity.

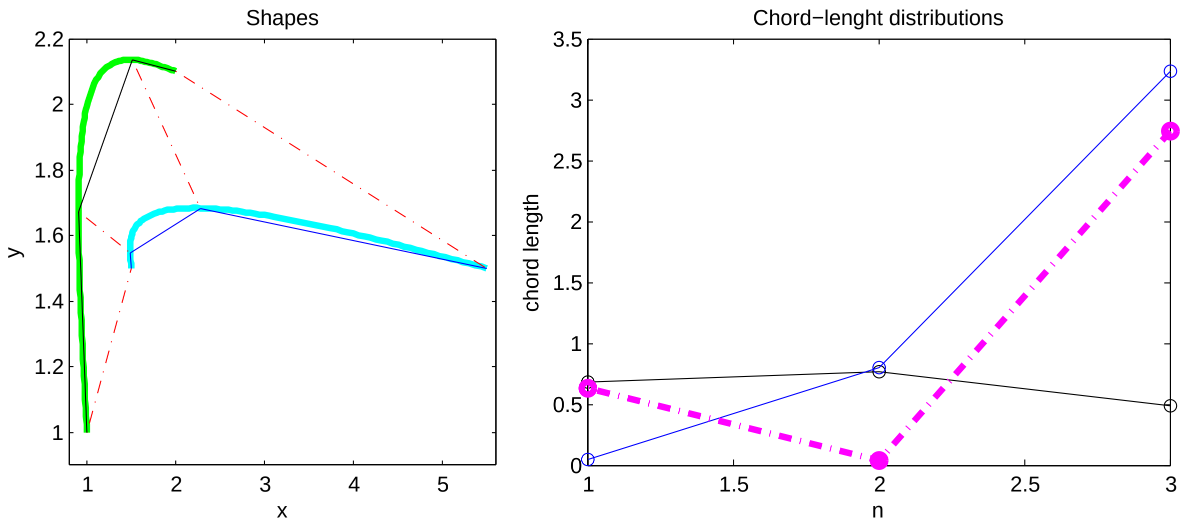


Figure 2.5: On the left: Four features per shape were chosen, with the red dashed lines showing their correspondence. The black and blue lines represent the inter-feature chords. On the right: The length of each of the three chords is plotted (circles). The magenta dashed line shows the absolute difference between the chord lengths.

## 2.2 State of the Art

### 2.2.1 Handwritten character parameterization, recognition and synthesis

Handwritten character parameterization, recognition and synthesis has been investigated with several methods. PCA based parameterization [6], DMPs framework [9] and a recurrent density networks technique [5] have been used to synthesise handwritten characters based on a set of example handwritten characters, however, no demonstration was done on the ability of the techniques to be used to classify an unknown character. An IOC ensemble method [21] demonstrated the ability to recognize and differentiate writing variations of the same character as well as synthesise new characters, in terms of capabilities this is the most similar to this work's parameterization which allows recognition and differentiation between different character variations, allows synthesising new characters with increased level of shape correctness (using several characters from a dataset) as well as characters with decreased level of correctness. This is achieved by a lossless (no information lost) parameterization consisting of path-tanget vectors' angle and magnitude. Path-tangent angle features have also been used with statistic models, i.e. HMM [13] and numerical algorithms, i.e. DTW [18], to parametrize handwritten characters in the field of signature recognition, however the usage differs in the way data was processed (angle wrapping and discretization) which the author argues leads to loss of vital information for the technique proposed in this work.

## 2.2.2 User interface

### 2.2.2.1 Interface paradigm

Teaching handwriting to a child with special needs has been done following one of two teaching approaches. Conventional teaching consisting of practise sessions under the guidelines of a 'master' (i.e. therapist) [7], or a more recent approach designated *Learning by Teaching paradigm* [10] which consists in having the pupil with difficulties, being the 'master' by teaching a 'less able' pupil (which might be another human or a machine) this can also be done under the supervision of a therapist. Each technique has its advantages/disadvantages. While conventional teaching might be more appropriate in the beginning, after the child shows a certain level of skill the use of *Learning by Teaching* allows to improve the children self-esteem and engagment in the activity [6], this work's framework allows the use of both teaching techniques with a computer partner to simulate either a 'master' or a 'pupil'. A third feature (it is unknown to the author if it exists in related works) is the possibility to produce reports on the performance of the user. These reports may be useful for the therapist involved.

### 2.2.2.2 Physical setup

An appropriate user interface is required to ensure the system captures the user input correctly while providing an intuitive and effortless user experience. Since the user trains by inputting characters in the system, that must be done in a similar way to handwriting, i.e. using a writing object in a 2D space, as any other setting would defeat the main purpose of handwriting training. Three conventional ways of digitalizing the user input are: Kinaesthetic, Computer vision and Digitizing pen. By kinaesthetic teaching a humanoid robot [2] is possible to obtain the end effector coordinates and therefore the pen-tip coordinates of the character. Tests were conducted using the humanoid robot available (Aldebaran's NAO), with motors set at zero torque. It was easy to conclude that while with some training effort a satisfactory calligraphy was achievable by an adult, this was not deemed a viable option. Using computer vision to digitalize the character written in paper, while attractive in a technical point of view, possibly providing more flexibility and robustness, is extremely time consuming to produce a solution with the mentioned attributes. This added complexity was deemed unnecessary to accomplish the core objectives of the work. The majority of the works in this area use handwritten characters captured online [5, 6, 9, 10, 13, 18, 21] (in this context **online** means the data is composed of sequences of pen tips usually captured via digitizing tablets, on the other hand **offline** means the character are available in images), this approach was also followed in this work.

# Chapter 3

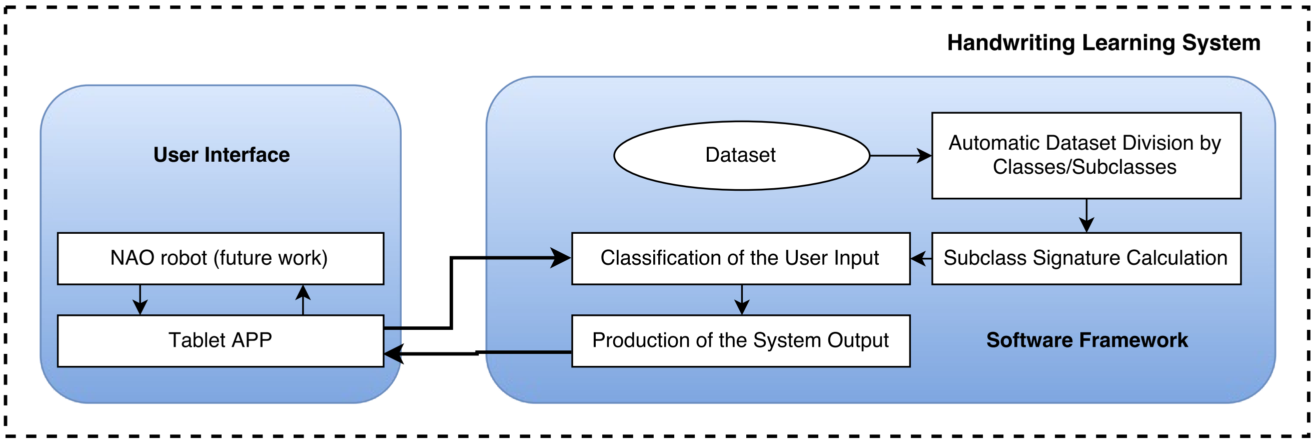
# Handwriting Learning: Proposed Method

## 3.1 Overview

This work requires a software framework able to accomplish the following tasks:

1. Automatic shape recognition;
2. Shape morphing of two characters;
3. Shape wise correctness evaluation of a character.

The framework described by the block diagram in figure 3.1 was devised to accomplish the objectives.



Handwriting Learning System
User Interface
NAO robot (future work)
Tablet APP
Dataset
Automatic Dataset Division by Classes/Subclasses
Classification of the User Input
Subclass Signature Calculation
Production of the System Output
Software Framework

Figure 3.1: Software framework diagram

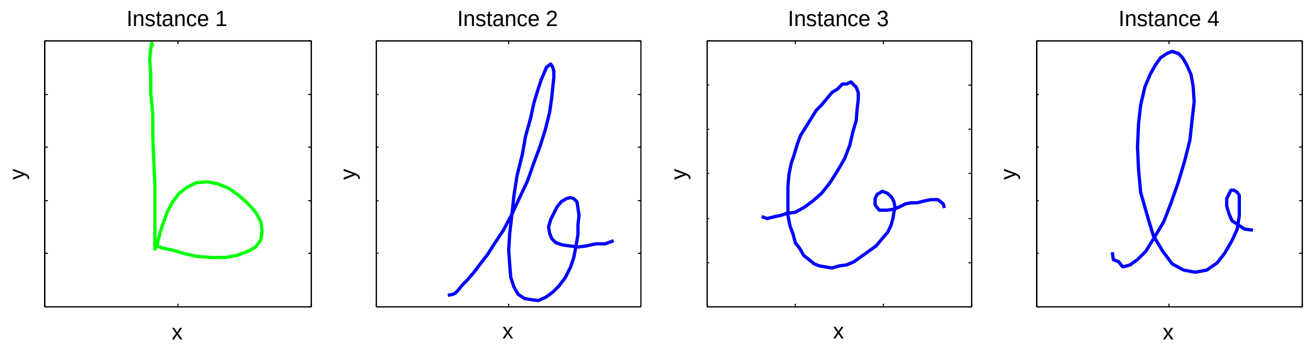


Figure 3.2: All the instances belong to the lower case 'b' class. Instances 2, 3 and 4 form one subclass, and instance 1 forms another subclass by virtue of being two different writing variations.

**Briefly examining the block diagram:** A dataset [11] is used to produce a **signature** for each character **subclass** (figure 3.2 exemplifies how subclasses are defined). This subclass signatures, called **master signatures**, represent all the characters from that subclass. The user input class is known, it is necessary to determine its subclass. The input signature is compared to the subclasses master signatures in order to automatically classify it. The system implements two learning paradigms with corresponding outputs:

1. An improved version of a character (produced by the system) using the user input as target, i.e., "Learning by Teaching".
2. An improved version of the user input using the master signature as target, i.e., "Conventional teaching".

In both learning cases, a report of the user difficulties can be produced by comparing the input with the master signature. Each block of the Software framework diagram (shown in figure 3.1) is explained individually in detail in the sections 3.2, 3.3, 3.4 and 3.5.

## 3.2 Automatic Dataset Division by Subclass

The dataset used, *UJI Pen Characters Data Set* [11], contains 120 instances of each character, captured using a digital pen (from 60 adult volunteers); Totalling 6240 instances for the ISO basic Latin alphabet alone. In the dataset all shapes are classified as to which character they correspond (division by class), however for the scope of this work that information is not enough. Therefore, the aim for this Section 3.2 is to automatically divide all the instances by subclasses (as defined in figure 3.2). The dataset contains characters comprised of 1, 2, 3 and 4 strokes.

To compare the characters, parameterizing their shapes in a convenient matter is necessary, so that the parameters are used as base of an effective comparison. Each character instance consists of a path described by its $x, y$ coordinates. A parameterization technique is proposed, which consists in converting the shapes' Cartesian coordinates to an 'Adapted Polar Coordinates'.

### 3.2.1 Adapted Polar Coordinates

For each shape path, defined by two coordinate vectors $x$ and $y$ (with $N$ elements), each pair $(x_i, y_i)$, $i = 1, \ldots, N$ defines one point of the path. The new parameterization of the shape consists of two vectors $\theta$ and $\rho$ (also with $N$ elements), calculated as show in the equation (3.1):

$$\begin{cases} \theta_i = \arctan\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right), & x_0 = 0 \wedge y_0 = 0, \quad i = 1, \ldots, N \\ \rho_i = \sum_{k=1}^{i} \sqrt{(x_k - x_{k-1})^2 - (y_k - y_{k-1})^2}, & x_0 = 0 \wedge y_0 = 0, \quad i = 1, \ldots, N \end{cases} \tag{3.1}$$

Resulting in the signature shown in figure 3.3. The vector $\theta$ is called the **signature** of the shape, and it is used as a pseudo-times series aligned using DTW to classify the shapes.

This parameterization has very important properties, namely:

*Property 1* - No information is lost when converting from $(x, y)$ to $(\theta, \rho)$ therefore the operation is reversible.

*Property 2* - Varying scale and position of a shape does not change its signature (vector $\theta$) with exception for $\theta_1$ that varies with position and scale.

*Property 3* - Each shape type has its characteristic signature (figure 3.4).

*Property 4* - The shape signature is a vector of angles, therefore every element of the vector is periodic with period $2\pi$ (or $360^o$) as shown in figure 3.5.

*Property 5* - A simple linear combination (after DTW) of two shapes' parameterization will result in a morphed shape parameterization, that can be used to calculate the $(x, y)$ coordinates of the morphed shape (Section 3.3).

When using $\rho$ for classification purposes (i.e. using chord-length distribution, Section 2.1.4), each character shape should be resized and moved, before parameterization, to fit a window defined by $0 < \mathbf{x} < 1 \wedge 0 < \mathbf{y} < 1$, an example on how to perform this operation is defined by Algorithm 2. This operation standardizes all shapes so that $\rho$ values can be compared.

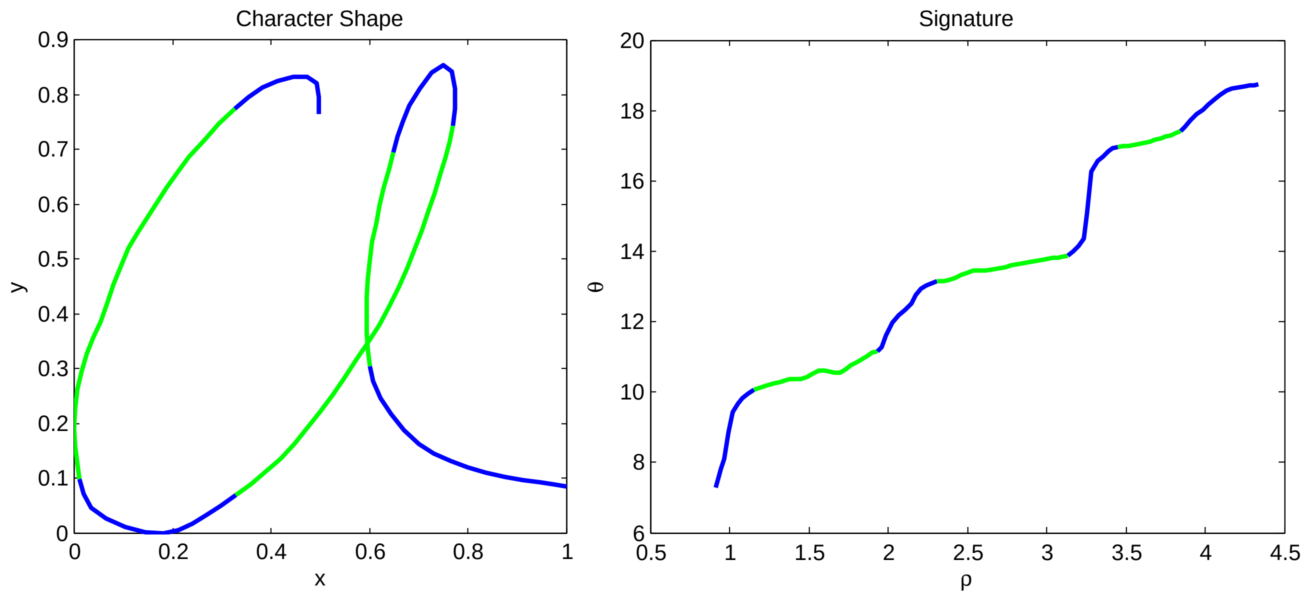


Figure 3.3: Example of a parameterization for a character 'a'. $\theta_i$ is the angle of the vector that points from point $(x_{i-1}, y_{i-1})$ to point $(x_i, y_i)$. $\rho_i$ is the length of the shape from point $(x_0, y_0)$ to point $(x_i, y_i)$. The blue sections of the shape have a large $\theta$ variation, therefore their signature is nearly vertical, on the other hand green sections are approximately straight lines therefore their signature is nearly horizontal.

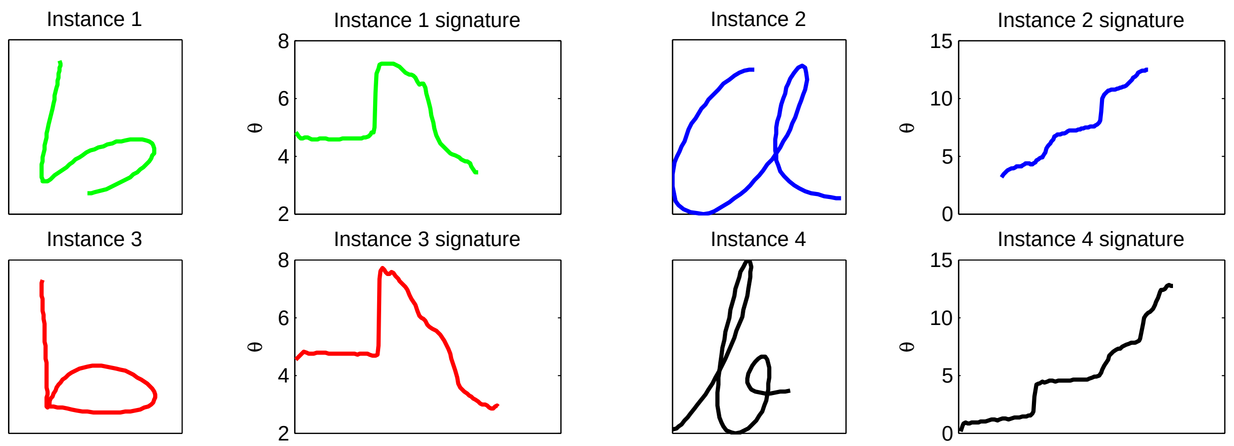


Figure 3.4: Instance 1 and 3 are of the same shape type (same subclass) and as expected their signatures are similar. On the other hand their signatures are different from instance 2 and 4 due to being of different shape types. The relevant features to evaluate similarity between signatures are: 1) Similar start and end value ($2\pi$ periodicity); 2) Same number of local maxima and minima with similar values ($2\pi$ periodicity).

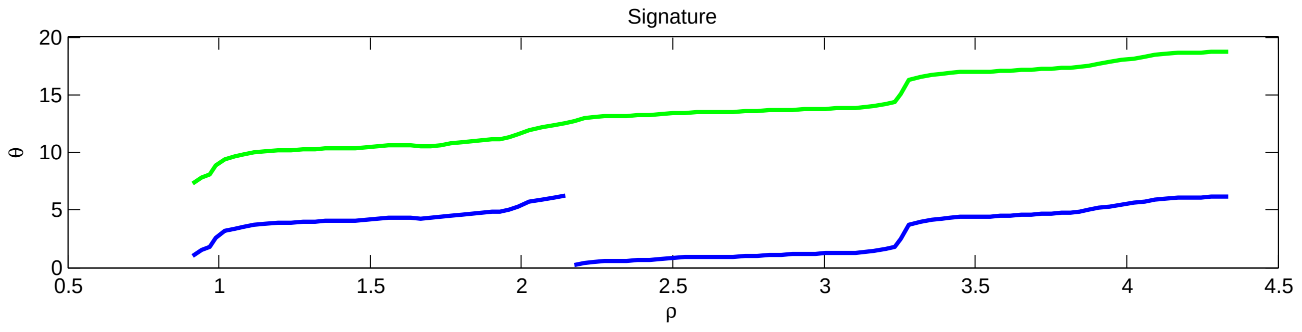


Figure 3.5: Blue and green signatures are completely equivalent. The blue signature is obtained by wrapping the green signature to a $[0, 2\pi]$ interval. In this work, the unwrapped version of a signature must always be used.

**Algorithm 2:** Shape standardisation

```
input  : A vector $x$
         A vector $y$
x=x-min(x) ;     // Subtracts the minimum of the vector x to all its elements
y=y-min(y);
maximum=max([x,y]) ;    // Calculates the biggest element in the concatenated vector
x=x/maximum ;                              // Divides all x elements by maximum
y=y/maximum;
output: A vector $x$
        A vector $y$
```

### 3.2.2 Signature Difference Matrix

The signatures defined in the previous section are directly correlated to the shape type because they are constructed by concatenating the angles of the vectors tangent at every point of the shape. Every character of the same type has a similar sequence of these angles (figure 3.3). By recognizing this property, one can use the signatures to divide a group of shapes in clusters of similar shapes.

Before attempting to classify the signatures, the first value of the signature ($\theta_1$) must be removed as defined by *Property 2* from Section 3.2.1 and a preprocessing of the signature is necessary to remove calligraphic serifs, swashes and tails from character shapes, as they contribute to additional $\theta$ variation that reduces the efficiency of the classification technique. This is done by considering that every calligraphic ornament is located either at the end or at the beginning of a stroke and consists of a

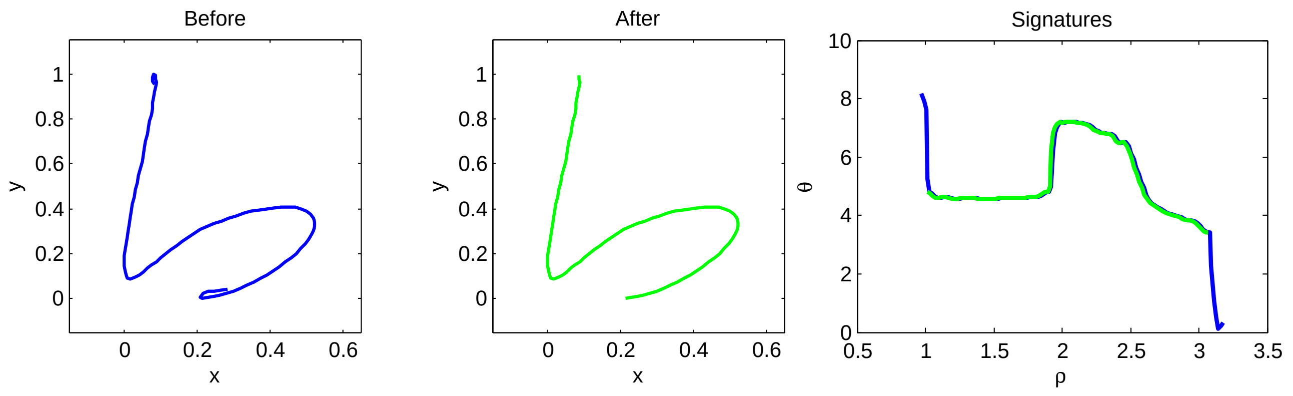


Figure 3.6: The blue character is the original and the green is the result of the pre-processing operation. It can be observed that the green signature has a smaller $\theta$ amplitude and only the beginning and end of the signature is affected.

section with large $\theta$ variation, as shown in figure 3.6. For shapes with multiple strokes this operation is performed individually in each stroke, after which all strokes signatures are concatenated into one.

As the name DTW implies, the algorithm was created to be used with time series, however, it can be used with any other kind of numeric sequence, i.e. ‘pseudo time series’. In this work the algorithm is used with numeric sequences composed of the angles of the vectors tangent to every point of a shape[1] (i.e. shape signatures). The angles are in radian units, as a consequence the pseudo time series actual values are periodic with period $2\pi$, therefore, to align two equivalent pseudo series it is necessary to consider the usual time dimension as well as this extra dimension (the variation of the signal values in $2\pi$ intervals). Using vectors **a** and **b** and matrix $AD$ defined in section 2.1.1 as example, this is achieved by varying **b** values by $2\pi k$, $k = -1, 0, 1$ thus creating three $AD$ matrices. The matrix with the smallest $AD_{n,m}$ value indicates de direction $k$ should vary to minimize the distance between the pseudo series. If the minimum corresponds to $k = 0$ the algorithm finishes. If it corresponds to $k = -1$, another $AD$ matrix is calculated for $k = -2$ and $AD_{n,m}$ is checked again. This process continues until $AD_{n,m}$ stops decreasing, the minimum $AD_{n,m}$ fixates the $k$ value for the optimal match between signals. An identical process is done if the minimum corresponds to $k = 1$ in the first iteration. After the minimum $k$ is found, the inspection for the optimal match path is done in the corresponding $AD$ matrix as defined for the conventional algorithm. Although a search in literature for this method of aligning periodic-valued pseudo series proved unfruitful thus suggesting it hasn’t been used before, the author doesn’t claim this as a new contribution because of it obvious nature.

[1]Ratanamahatana [14] suggests using DTW to align pseudo time series composed of angles in the context of feature matching between images.

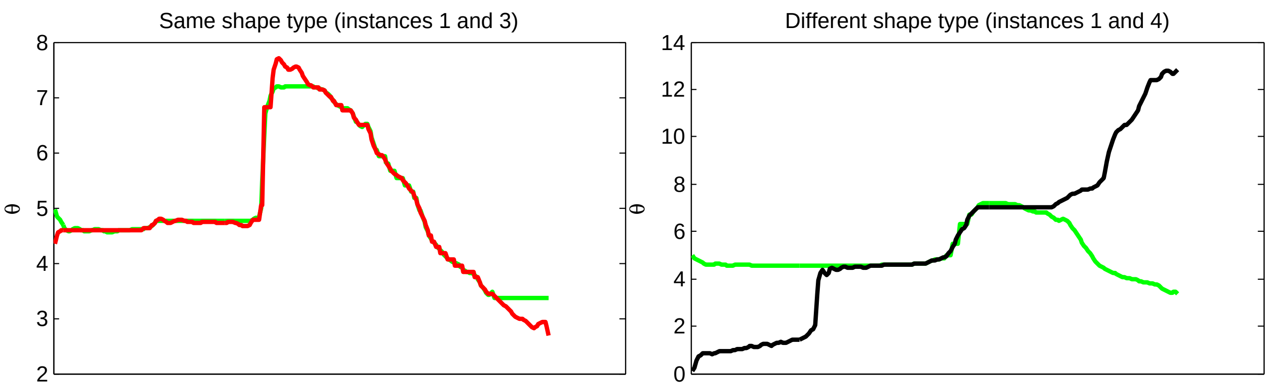


Figure 3.7: Applying the DTW operation to the signatures from figure 3.4, it is possible to calculate the $aaad = 0.0894$ for the signatures in the left graph and $aaad = 2.2880$ for the right. Since this value indicates a difference, the larger its magnitude, the larger the difference between the shapes.

For this Modified DTW algorithm the function prototype used throughout this work follows the following form, *[$a_{path}$, $b_{path}$, k] = modedDTW(a, b)* .

The modified DTW algorithm, is used to calculate the optimal match between the two signatures (pseudo-time series) $\theta 1$ and $\theta 2$ by warping them non-linearly and incrementing by $2\pi$ one in relation to the other. After the DTW operation both signatures have the same number of elements ($n$ elements) and they are said to be **aligned** as shown in figure 3.7, this can be considered as a process of 'continuous feature matching'. For two signatures $\theta 1$ and $\theta 2$ a measure of similarity between the aligned signatures, called average absolute amplitude difference (aaad), was created and defined in equation (3.2). This similarity measure can only be used to compare signatures that have been previously aligned.

The *aaad* is computed as follows:

$$aaad = \frac{\sum_{i=1}^{n} |\theta 1_i - \theta 2_i|}{n} \tag{3.2}$$

The operation is repeated for every pair combination of the dataset to be classified. For a dataset with $k$ character instances, a $k \times k$ *Signature Difference Matrix (SDM)* is created. The element $sdm_{ij}$ stores the *aaad* between the instance $\theta i$ and instance $\theta j$. As an example, the operation was executed in 20 instances of 'b' character, shown in figure 3.8 resulting in the *SDM* shown in table 3.1. The *SDM* is always symmetric with diagonal values equal to zero.

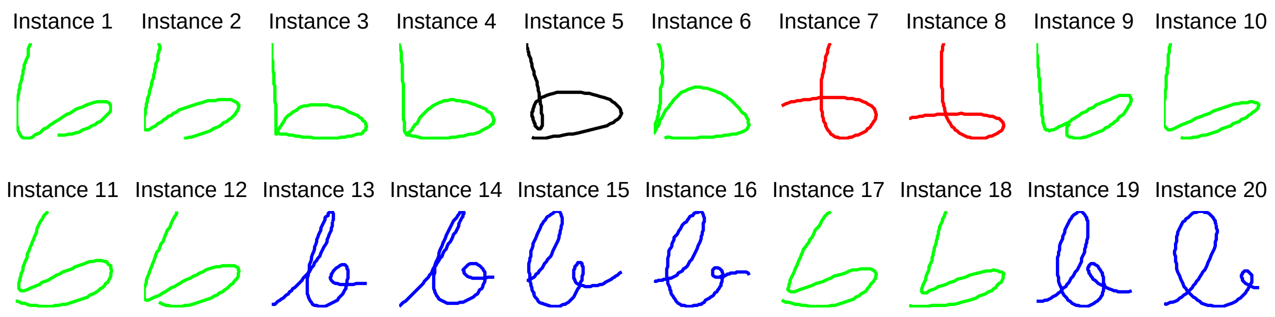


Figure 3.8: Class formed by randomly choosen instances of ‘b’ character of the dataset. Each colour represents a subclass.

Table 3.1: SDM for the 20 instances of ’b’ character shown in figure 3.8

| **Inst.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0.04 | 0.08 | 0.07 | 2.03 | 0.10 | 1.23 | 1.22 | 0.13 | 0.06 | 0.08 | 0.05 | 2.21 | 2.09 | 2.22 | 2.13 | 0.09 | 0.04 | 2.21 | 1.85 |
| **2** | 0.04 | 0 | 0.11 | 0.11 | 2.02 | 0.11 | 1.30 | 1.27 | 0.15 | 0.06 | 0.11 | 0.07 | 2.47 | 2.29 | 2.50 | 2.35 | 0.12 | 0.05 | 2.45 | 2.03 |
| **3** | 0.08 | 0.11 | 0 | 0.04 | 1.84 | 0.05 | 1.49 | 1.54 | 0.14 | 0.09 | 0.09 | 0.07 | 2.24 | 2.16 | 2.40 | 2.21 | 0.09 | 0.06 | 2.27 | 1.94 |
| **4** | 0.07 | 0.11 | 0.04 | 0 | 1.78 | 0.06 | 1.56 | 1.60 | 0.12 | 0.09 | 0.12 | 0.09 | 2.39 | 2.28 | 2.55 | 2.30 | 0.13 | 0.09 | 2.42 | 2.07 |
| **5** | 2.03 | 2.02 | 1.84 | 1.78 | 0 | 1.80 | 4.38 | 4.30 | 1.02 | 2.00 | 1.97 | 1.89 | 5.25 | 5.20 | 5.12 | 5.01 | 1.98 | 1.99 | 5.17 | 5.03 |
| **6** | 0.10 | 0.11 | 0.05 | 0.06 | 1.80 | 0 | 1.55 | 1.56 | 0.20 | 0.15 | 0.17 | 0.12 | 2.60 | 2.56 | 2.59 | 2.67 | 0.17 | 0.11 | 2.63 | 2.35 |
| **7** | 1.23 | 1.30 | 1.49 | 1.56 | 4.38 | 1.55 | 0 | 0.03 | 1.57 | 1.42 | 1.58 | 1.46 | 0.98 | 0.96 | 0.98 | 0.99 | 1.56 | 1.47 | 1.01 | 0.87 |
| **8** | 1.22 | 1.27 | 1.54 | 1.60 | 4.30 | 1.56 | 0.03 | 0 | 1.59 | 1.41 | 1.51 | 1.36 | 0.99 | 0.95 | 1.02 | 1.03 | 1.47 | 1.41 | 1.02 | 0.91 |
| **9** | 0.13 | 0.15 | 0.14 | 0.12 | 1.02 | 0.20 | 1.57 | 1.59 | 0 | 0.09 | 0.08 | 0.08 | 2.31 | 2.18 | 2.52 | 2.23 | 0.10 | 0.11 | 2.36 | 1.99 |
| **10** | 0.06 | 0.06 | 0.09 | 0.09 | 2.00 | 0.15 | 1.42 | 1.41 | 0.09 | 0 | 0.04 | 0.05 | 2.26 | 2.13 | 2.35 | 2.21 | 0.04 | 0.04 | 2.26 | 1.87 |
| **11** | 0.08 | 0.11 | 0.09 | 0.12 | 1.97 | 0.17 | 1.58 | 1.51 | 0.08 | 0.04 | 0 | 0.05 | 2.36 | 2.21 | 2.50 | 2.15 | 0.04 | 0.05 | 2.28 | 1.88 |
| **12** | 0.05 | 0.07 | 0.07 | 0.09 | 1.89 | 0.12 | 1.46 | 1.36 | 0.08 | 0.05 | 0.05 | 0 | 2.48 | 2.24 | 2.49 | 2.21 | 0.06 | 0.04 | 2.34 | 1.93 |
| **13** | 2.21 | 2.47 | 2.24 | 2.39 | 5.25 | 2.60 | 0.98 | 0.99 | 2.31 | 2.26 | 2.36 | 2.48 | 0 | 0.04 | 0.06 | 0.07 | 2.50 | 2.30 | 0.04 | 0.06 |
| **14** | 2.09 | 2.29 | 2.16 | 2.28 | 5.20 | 2.56 | 0.96 | 0.95 | 2.18 | 2.13 | 2.21 | 2.24 | 0.04 | 0 | 0.07 | 0.10 | 2.34 | 2.17 | 0.07 | 0.07 |
| **15** | 2.22 | 2.50 | 2.40 | 2.55 | 5.12 | 2.59 | 0.98 | 1.02 | 2.52 | 2.35 | 2.50 | 2.49 | 0.06 | 0.07 | 0 | 0.08 | 2.52 | 2.39 | 0.05 | 0.08 |
| **16** | 2.13 | 2.35 | 2.21 | 2.30 | 5.01 | 2.67 | 0.99 | 1.03 | 2.23 | 2.21 | 2.15 | 2.21 | 0.07 | 0.10 | 0.08 | 0 | 2.23 | 2.18 | 0.06 | 0.08 |
| **17** | 0.09 | 0.12 | 0.09 | 0.13 | 1.98 | 0.17 | 1.56 | 1.47 | 0.10 | 0.04 | 0.04 | 0.06 | 2.50 | 2.34 | 2.52 | 2.23 | 0 | 0.05 | 2.40 | 1.96 |
| **18** | 0.04 | 0.05 | 0.06 | 0.09 | 1.99 | 0.11 | 1.47 | 1.41 | 0.11 | 0.04 | 0.05 | 0.04 | 2.30 | 2.17 | 2.39 | 2.18 | 0.05 | 0 | 2.33 | 1.90 |
| **19** | 2.21 | 2.45 | 2.27 | 2.42 | 5.17 | 2.63 | 1.01 | 1.02 | 2.36 | 2.26 | 2.28 | 2.34 | 0.04 | 0.07 | 0.05 | 0.06 | 2.40 | 2.33 | 0 | 0.04 |
| **20** | 1.85 | 2.03 | 1.94 | 2.07 | 5.03 | 2.35 | 0.87 | 0.91 | 1.99 | 1.87 | 1.88 | 1.93 | 0.06 | 0.07 | 0.08 | 0.08 | 1.96 | 1.90 | 0.04 | 0 |

### 3.2.3 Division by Subclass

The *SDM* is a distance matrix because its elements are a measure of the ‘distance’ between two shapes (a higher value means more differences). Inside the *SDM* there are groups of shapes with relatively low distance between all elements of the group, they form a subclass cluster (same shape type). A clustering algorithm could be used to calculate those groups, however conventional clustering algorithms (like k-means, Expectation-Maximization (EM), DBSCAN, OPTICS) are more suitable to cluster data described by its absolute coordinates. The *SDM* data is described by their relative coordinates (or relationship between elements) therefore requiring modifications in conventional clustering algorithms to be used.

A breakthrough is achieved if a element-wise exponentiation of the *SDM* matrix is performed, i.e. $sam_{ij} = sdm_{ij}^{-1}$. Care should be taken with the diagonal elements, before the operation, their magnitude should be recalculated to be equal or smaller than the smallest non-zero element of *SDM* (this operation assures the *self-loop weigth*, as explained in section 2.1.3, is not $\approx Infinite$ which would cause extreme cluster granularity and defeat the process of division by subclass). The new matrix, *SAM*, is an adjacency matrix, widely used in computer science to represent a finite graph, therefore allowing the use of a graph based clustering algorithm. The Markov Cluster Algorithm (MCL)[19], explained in section 2.1.3, is used to cluster the *SAM*.

As an example, the operation described in this section was applied to the SDM in table 3.1 resulting in the following clustering $\{\{5\}; \{7,8\}; \{13, 14, 15, 16, 19, 20\}; \{1,2,3,4,6,9,10,11,12,17,18\}\}$, it can be verified by checking figure 3.8 that the classification was 100% successful. For this example each cluster represents a subclass of the character ‘b’, i.e. a variation in the calligraphic font.

## 3.3 Subclass Signature Calculation

As stated in *Property 5* from section 3.2.1, a linear combination of the aligned APC parameterizations of two characters creates a new parameterization defining a new shape. Varying the coefficients of the linear combination allows to vary how much each each input shape affects the resulting shape. This process **APC parameterization + Modified DTW + Linear combination** defines a novel[2] shape morphing method. Using this method all characters of a cluster can be morphed into a single shape, called master character. For a cluster with $n$ characters each with $s$ strokes, the $i^{th}$ stroke of the $k^{th}$

[2]The author is unaware of any previous utilization of this technique.

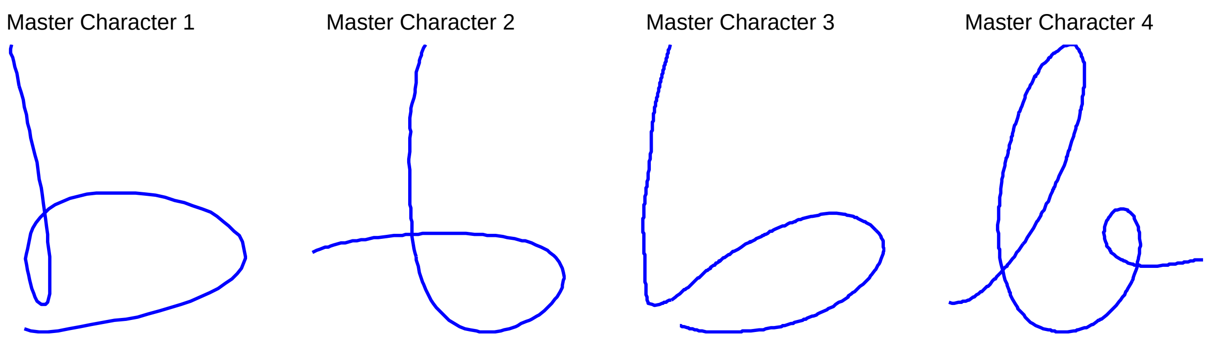


Figure 3.9: Master characters for the subclasses contained in the class defined in figure 3.8. Master 1 corresponds to cluster containing shape 5, Master 2 to cluster {7,8}, Master 3 to {1,2,3,4,6,9,10,11,12,17,18} and Master 4 to {13, 14, 15, 16, 19, 20}.

character has the APC parameterization $(\theta k^i, \rho k^i)$, $k = 1,\ldots,n$; $i = 1,\ldots,s$. The $i^{th}$ stroke of the master character has the APC parameterization $(\theta M^i, \rho M^i)$, in which $\theta M$ is called master signature. The algorithm 3 defines the process to obtain $(\theta M^i, \rho M^i)$.

This operation has several interesting properties, while it condenses all the instances information into a single one, by doing so also intrinsically improves the character shape correctness. This can be proved using the 'signal averaging' technique from signal processing. For $n$ characters of the same subclass, if every character signature $\theta_i(k)$ $(i = 1,\ldots,n)$ is considered an independent measurement of a perfect character signature $\theta P(k)$. Can be considered that each $\theta_i(k)$ will be composed of a perfect character signature plus an uncorrelated gaussian noise signal $z_i(k)$ (in this context noise is shape deformities) $\theta_i(k) = \theta P(k) + z_i(k)$. Calculating the master signature $\theta M(k)$ by averaging all character signatures increases the signal to noise ratio ($SNR$[3]) by $n$ as proved in equation (3.3). Therefore the larger the $n$ (number characters of the same subclass) the better is the approximation of the master character to a perfect character:

$$\begin{aligned} SNR_i(k) &= \frac{E(|\theta P(k)|^2)}{E(|z_i(k)|^2)} \\ SNR_M(k) &= \frac{E(|\frac{\sum_{i=1}^{n}\theta P(k)}{n}|^2)}{E(|\frac{\sum_{i=1}^{n} z_i(k)}{n}|^2)} = n\frac{E(|\theta P(k)|^2)}{E(|z(k)|^2)} \end{aligned} \tag{3.3}$$

As an example, the operation described in this section was applied to the clusters defined in section 3.2.3 resulting in the shapes showed in figure 3.9.

[3]$SNR_i$ - the signal to noise ratio of the signature i and $SNR_M$ - the signal to noise ratio of the master averaged signature

## 3.4 Classification of the User Input

Using the techniques described in section 3.2, the full dataset can be divided into subclasses, with each subclass being described by: One master parameterization; An ASCII code; The number of strokes. Using the user input APC parameterization, obtained by applying the tool described in section 3.2.1, a comparison with all the subclasses can be done so that a probability (and a standard deviation for that probability) can be calculated to whether the user input belongs to a certain subclass. For a user input described by the APC parameters $(\theta U, \rho U,)$ and a subclass described by $(\theta M, \rho M,)$ the pseudo-algoritmh 4 describes how the probability and probability standard deviation is calculated of the user input belonging to that class (before applying the algorithm is applied both the user input's and the subclass signatures' APC strokes should be concatenated). This information allows not only to determine what subclass the user input belong but also allows to classify the user input in term of shape correctness, by defining a minimum threshold is defined (for probability and probability standard deviation) where above it the letter shape is considered satisfactory.

## 3.5 Output Production

As referred in section 3.1 the system output depends in which mode the software is running.

1. When simulating that the system is learning from the user input. The system firstly outputs a character with some purposively created deformities (procidure explained in section 3.5.1), then the user suggests an improvement. The next output of the system is a character created using the morphing technique explained in section 3.3, in which the base character is the system previous character and the target character is the user suggestion. The linear combination parameters can be varied to simulate a faster or slower learning speed.
2. When the system is teaching the user, the output of the system is a character created using the morphing technique explained in section 3.3, in which the base character is the user's input and the target character is the master character of the subclass the user input belongs. The linear combination parameters can be varied to define a faster or slower teaching speed.

In both learning cases, a report of the user difficulties can be produced. The user character signature is compared to the master signatures of their respective classes and the points where exists bigger difference are highlighted. This allows to highlight the parts of the user character that are furthest from the correct shape and therefore should be improved.

### 3.5.1 Low Calligraphic Correctness Characters Generation

As explained in section 1 the ‘Learning by teaching’ paradigm consists in having a student with difficulties teaching another student with more difficulties. In order to simulate the student with bigger writing difficulties, the system must generate characters with low calligraphic correctness. This can be easily achieve with the APC parameterization since a set of random perturbations to the $\theta$ and $\rho$ vectors will induce shape deformations. Many different ways of generating and adding the perturbations can be studied. The author suggests a simple but effective way. For a $(\theta, \rho)$ parameterization of a stroke with $n$ points. Using two random coefficients $coef_\theta \in [-1, 1]$ and $coef_\rho \in [-1, 1]$, two vectors $\theta_\xi$ and $\rho_\xi$ are generated as shown in equation (3.4).

$$\begin{cases} \theta_\xi & = \{1 + coef_\theta \frac{2k-n-1}{n-1}, k \in \{0, \ldots, n-1\}\} \\ \rho_\xi & = \{1 + coef_\rho \frac{2k-n-1}{n-1}, k \in \{0, \ldots, n-1\}\} \end{cases} \tag{3.4}$$

The parameterization of the new stroke will be $(\theta + \theta_\xi, \rho + \rho_\xi)$ and by varying $coef_\theta$ and $coef_\rho$ different shape deformations can be achieved. For a letter with several strokes this should be done to each one individually with the same $coef_\theta$ and $coef_\rho$ parameters.

**Algorithm 3:** Subclass Signature Calculation

```
input  : Number of characters in cluster $n$
         Number of strokes per character $s$
         APC parameterization of all cluster's characters $(\theta -^{-}, \rho -^{-})$, $n \times 2 \times s$ matrix of arrays
for $i$=1 to $s$ do
    // The parameterization of the first character is stored in the master
       parameterization
    $\theta M^i = \theta 1^i$;
    $\rho M^i = \rho 1^i$;
    for $k$=2 to $n$ do
        // This function operates as described in section 2.1.1
        [path1, path2, $twoPIverticalDisplacement$] = modedDTW($\theta M^i$, $\theta k^i$);

        // Warping parameterizations according to path1 and path2 vectors and
           moving the signature vertically according to
           $twoPIverticalDisplacement$ scalar
        $\theta M^i = \theta M^i(path1)$;
        $\theta k^i = \theta k^i(path2) + 2\pi \times twoPIverticalDisplacement$;
        $\rho M^i = \rho M^i(path1)$;
        $\rho k^i = \rho k^i(path2)$;

        // This linear combination coefficients produce the average of all
           parameterizations
        $\theta M^i = \theta M^i \times (1 - \frac{1}{k}) + \theta k^i \times \frac{1}{k}$ ;   // Care should be taken when summing $\theta$'s,
        check appended code
        $\rho M^i = \rho M^i \times (1 - \frac{1}{k}) + \rho k^i \times \frac{1}{k}$;
    end
end
output: APC parameterization of cluster's master character $(\theta M^{-}, \rho M^{-})$, $1 \times 2 \times s$ matrix of
        arrays
```

**Algorithm 4:** User input classification

```
input  : APC parameterization of user input $(\theta U, \rho U)$, $1 \times 2$ matrix of arrays
// This function operates as described in section 2.1.1
[path1, path2, $twoPIverticalDisplacement$] = modedDTW($\theta M$, $\theta U$);

// Warping parameterizations according to path1 and path2 vectors and
   moving the signature vertically according to $twoPIverticalDisplacement$
   scalar
$\theta M = \theta M(path1)$;
$\theta U = \theta U(path2) + 2\pi \times twoPIverticalDisplacement$;
$\rho M = \rho M(path1)$;
$\rho U = \rho U(path2)$;

// Subtract element by element (vector) -> Absolut values (vector) -> Sums
   all values (Scalar) -> Sums 1 (Scalar) -> Exponentiate to -2 (Scalar) ->
   Multiplies by 100 (Scalar)
$probability = 100 \times (sum(abs(\theta M - \theta U)) + 1)^{-2}$;
$standarddeviation = 10 * sum(abs(\rho M - \rho U))$;
output: APC parameterization of subclasses $(\theta M, \rho M)$, $1 \times 2$ matrix of arrays
```

# Chapter 4

# Experiments, Tests and Results

In this chapter, the performance of the proposed system is assessed and validated. First, the results of the automatic classification of the training dataset (*UJI Pen Characters Data Set* [11]) are shown along with the result of the morphing techique created. Then, the system is tested with simulated user inputs, using the publicly available *Chars74K dataset*[4] as testing dataset. The author believes the use of a training dataset and a testing dataset from two different sources allows to further prove the robustness of the system.

## 4.1 Automatic Dataset Classification

Using the techniques described in sections 3.2 and 3.3 the training dataset was divided in subclasses and the master character for each one was calculated. Tables 4.1 and 4.2 show how many instances per letter were available in the training dataset and how many subclasses were created (tables with this informations for all letters are accessible in the appendixes).

Some letters have few writing variations, this leads to the creation of few subclasses with a large number of instances each. As a consequence, for these types of letters if a subclass does not have a large number of instances it is usually constituted of only one or two instances, which are, very likely a badly written character (for subclasses with one instance) or a very rare writing variation (for subclasses with two instances) as shown in figures 4.1, 4.2 and 4.3. In tables 4.1 and 4.2 the letters for which fewer subclasses were created are, are examples of the previously described: **C**, **c**, **e**, **g**, **O**, **o** and **Q** written with 1 stroke, also **B**, **D** and **P** written with 2 strokes and **E** written 4 strokes.

On the other hand some letters have many writing variations, although some are very similar the

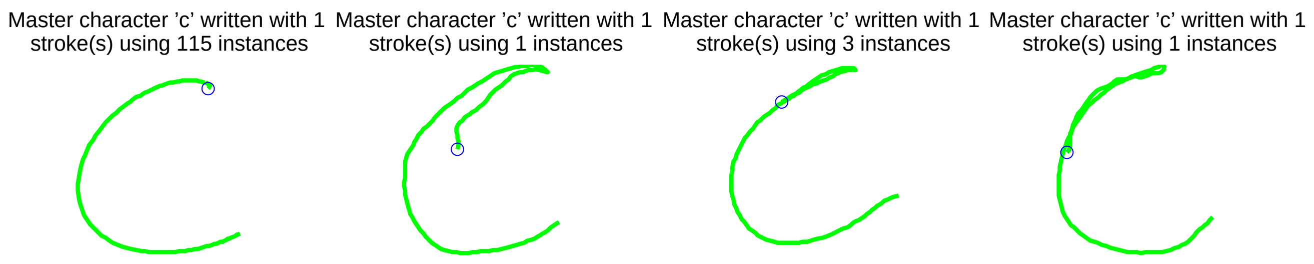

Master character 'c' written with 1 stroke(s) using 115 instances
Master character 'c' written with 1 stroke(s) using 1 instances
Master character 'c' written with 1 stroke(s) using 3 instances
Master character 'c' written with 1 stroke(s) using 1 instances

Figure 4.1: Example of a letter with few writing variations. For lower-case 'c' written with 1 stroke, only four subclasses have been created (check table 4.2), of these four, only the first and third subclasses are relevant. The instances in the other subclasses have been correctly identified has outliers and therefore isolated in their own subclasses.

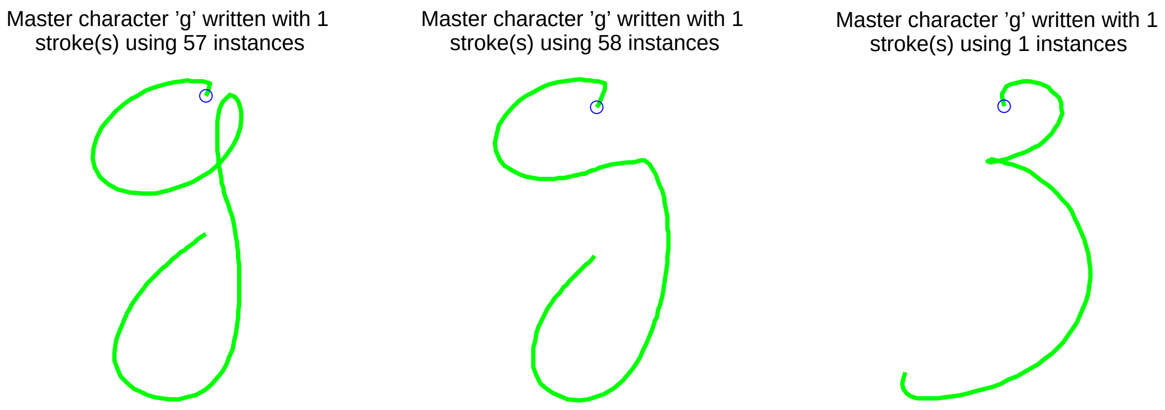

Master character 'g' written with 1 stroke(s) using 57 instances
Master character 'g' written with 1 stroke(s) using 58 instances
Master character 'g' written with 1 stroke(s) using 1 instances

Figure 4.2: Example of a letter with few writing variations. For lower-case 'g' written with 1 stroke, only three subclasses have been created (check table 4.2), of these three, only the first two subclasses are relevant. The instance in the third subclass has been correctly identified has outlier and therefore isolated in one subclasses.

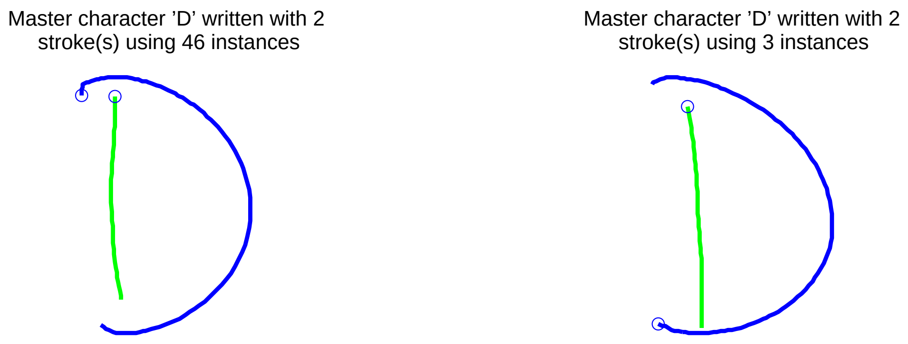

Master character 'D' written with 2 stroke(s) using 46 instances
Master character 'D' written with 2 stroke(s) using 3 instances

Figure 4.3: Example of a letter with few writing variations. For upper-case 'D' written with 2 stroke, only two subclasses have been created (check table 4.2) both have three or more instances therefore both can be considered relevant. The first stroke is in green and the second in blue. As a side note, by experience the author believes the second subclass elements have been drawn by left handed individuals.

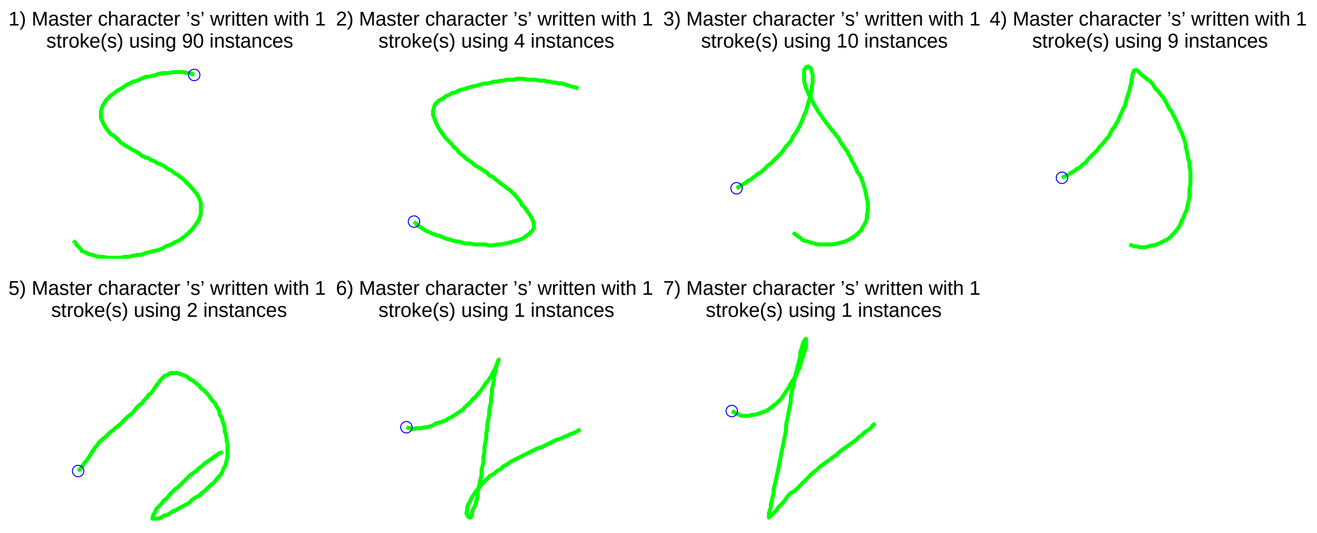
system seems to correctly separates them in different subclasses. Examples are shown in figures 4.4 and 4.5. In tables 4.1 and 4.2 the letters with many subclasses created, are examples of the described: **B**, **b**, **D** and **s** written with 1 stroke, also **A** and **Q** written with 2 strokes.

Since user input classification is done in real time, having few subclasses to compare against enhances the responsiveness of the application. Considering that having fewer subclasses implies having more instances per subclass and higher number of instances per subclass produces higher quality master characters, as proved in equation3.3 it is of paramount importance that as most instances as possible are agglomerated in their respective subclasses. Figure 4.6 shows the distribution of instances per class size, is it possible to conclude that subclasses with around five or more instances concentrate 87.55% of the instances, this translates in a significant decrease in user input classification time and a significant increase in master character shape quality.

As a *near extensive* example of the resulting parameterizations the master characters of the biggest 100 subclasses, which represent 75.45% of the training dataset, are shown in figure 4.7.

## 4.2 Letter Recognition

Several approaches could be used to achieve a systematic testing of the recognition system. A long period of testing with real users could meet this goal, this approach however would be very time consuming and logistically expensive. A faster and as effective alternative can be using a testing dataset. Given the time frame available this methodology was preferred. The *Chars74K dataset*[4] was used.



1) Master character 's' written with 1 stroke(s) using 90 instances
2) Master character 's' written with 1 stroke(s) using 4 instances
3) Master character 's' written with 1 stroke(s) using 10 instances
4) Master character 's' written with 1 stroke(s) using 9 instances
5) Master character 's' written with 1 stroke(s) using 2 instances
6) Master character 's' written with 1 stroke(s) using 1 instances
7) Master character 's' written with 1 stroke(s) using 1 instances

Figure 4.4: Example of a letter with several writing variations. For lower-case 's' written with 1 stroke, the first four subclasses have three or more instances therefore can be considered relevant. Others are most probably outliers or rare writing variations. It is important to observe that the classification system considers subclass 3 and 4 are two different writing variations due to the existence/absence of the top loop. A testimony to the system effectiveness is how subclass 1 and 2 despite having equal final shapes, are considered different writing variations due to being the result of two different writing procedures.

Table 4.1: Number of existing instances per letter and stroke

| | Strokes | | | |
|---|---|---|---|---|
| Character | 1 | 2 | 3 | 4 |
| A | 21 | 94 | 5 | 0 |
| B | 69 | 50 | 1 | 0 |
| C | 120 | 0 | 0 | 0 |
| D | 71 | 49 | 0 | 0 |
| E | 14 | 17 | 85 | 4 |
| O | 120 | 0 | 0 | 0 |
| P | 60 | 60 | 0 | 0 |
| Q | 5 | 115 | 0 | 0 |
| c | 120 | 0 | 0 | 0 |
| e | 120 | 0 | 0 | 0 |
| g | 116 | 4 | 0 | 0 |
| o | 120 | 0 | 0 | 0 |
| s | 117 | 3 | 0 | 0 |

Table 4.2: Number of subclasses created per letter and stroke

| | Strokes | | | |
|---|---|---|---|---|
| Character | 1 | 2 | 3 | 4 |
| A | 10 | 10 | 2 | 0 |
| B | 9 | 2 | 1 | 0 |
| C | 6 | 0 | 0 | 0 |
| D | 13 | 2 | 0 | 0 |
| E | 8 | 6 | 11 | 1 |
| O | 4 | 0 | 0 | 0 |
| P | 8 | 6 | 0 | 0 |
| Q | 1 | 5 | 0 | 0 |
| c | 4 | 0 | 0 | 0 |
| e | 1 | 0 | 0 | 0 |
| g | 3 | 3 | 0 | 0 |
| o | 7 | 0 | 0 | 0 |
| s | 7 | 2 | 0 | 0 |

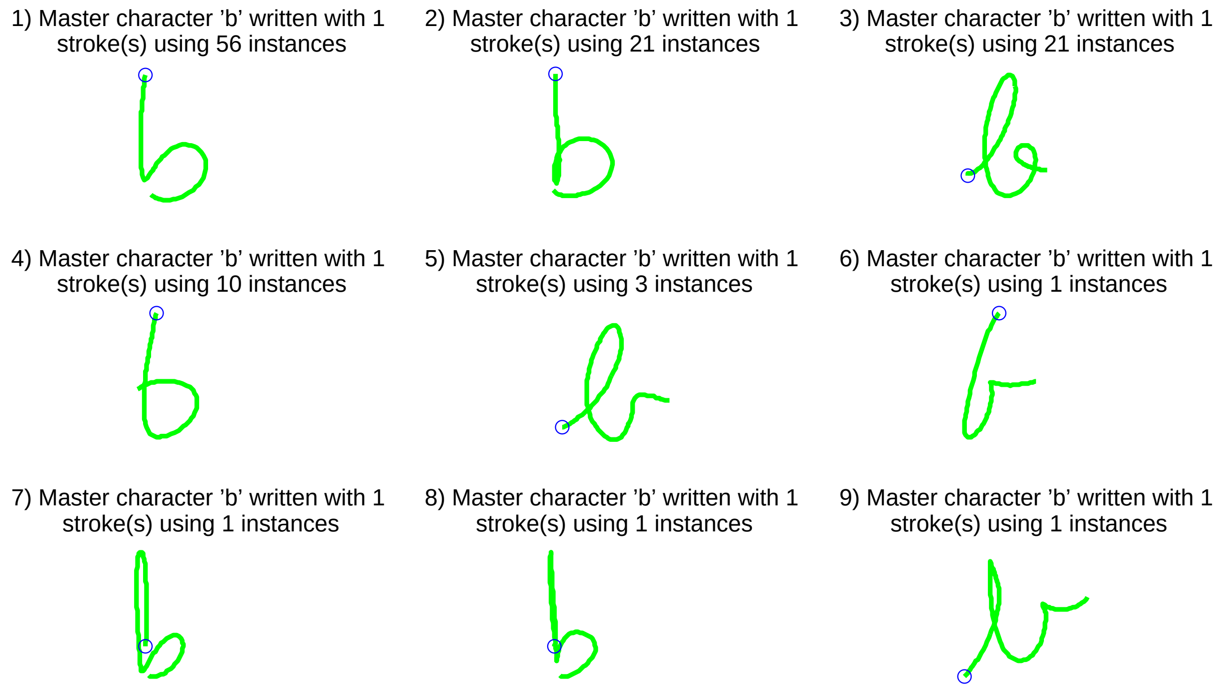

1) Master character 'b' written with 1 stroke(s) using 56 instances
2) Master character 'b' written with 1 stroke(s) using 21 instances
3) Master character 'b' written with 1 stroke(s) using 21 instances
4) Master character 'b' written with 1 stroke(s) using 10 instances
5) Master character 'b' written with 1 stroke(s) using 3 instances
6) Master character 'b' written with 1 stroke(s) using 1 instances
7) Master character 'b' written with 1 stroke(s) using 1 instances
8) Master character 'b' written with 1 stroke(s) using 1 instances
9) Master character 'b' written with 1 stroke(s) using 1 instances

Figure 4.5: Example of a letter with several writing variations. For lower-case 'b' written with 1 stroke, the first five subclasses have three or more instances therefore can be considered relevant. Others are most probably outliers or rare writing variations. It is arguable that subclasses 5 and 9 might be the same writing variation, although by analysing the inner working of the classification one can conclude that the curvature of the last part (one is concave up, the other concave down) of the letters is what separated them is different classes.

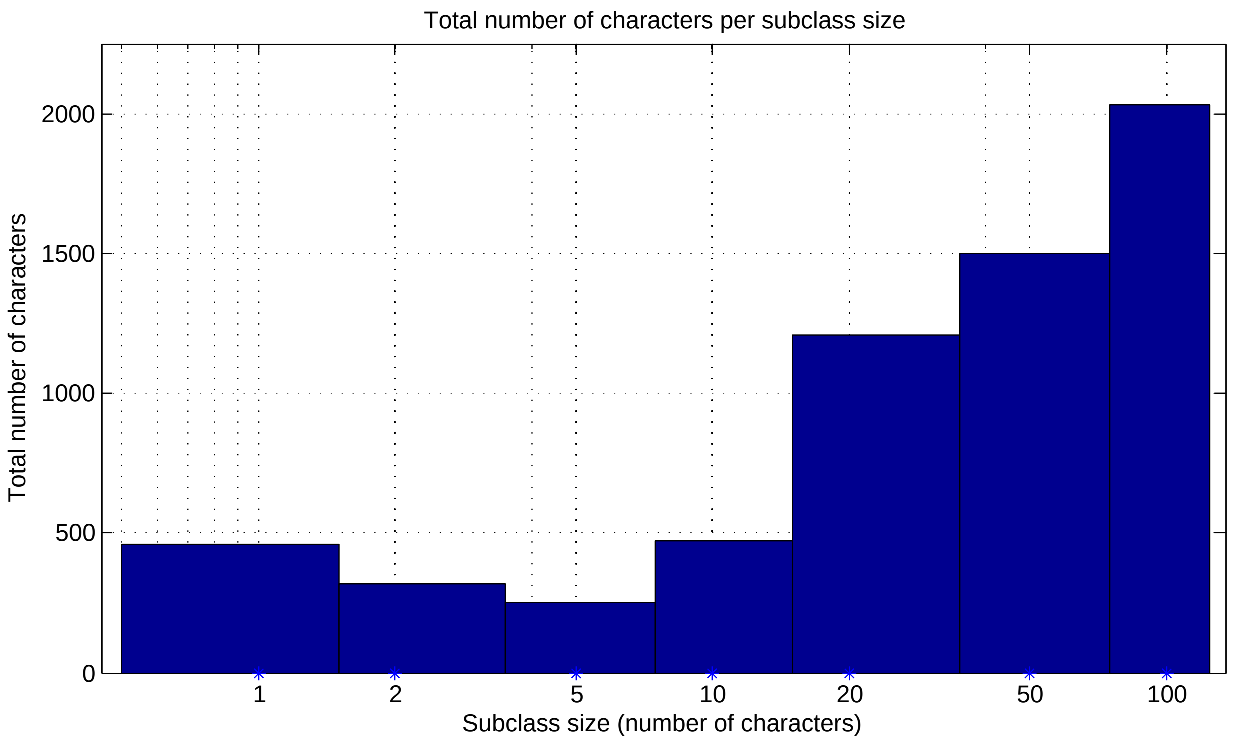


Figure 4.6: Histogram with the distribution of instances per subclass. Subclasses with just one instances totalize just under 500 instances. Exact bin sizes are {458, 319, 251, 472, 1210, 1500, 2030}.

The testing dataset contains 55 instances per letter for a total of 2860, of those:

- 31 instances are non classifiable because they belong to classes (letter+strokes) that are empty in the training dataset;
- 389 instances belong to classes that have more instances in the testing dataset than in the training dataset (the training dataset has 200 instances for these classes), these will be designated *under-represented instances*;
- 2440 instances belong to classes in the testing dataset of equal or smaller size when comparing with the training dataset (the training dataset has 6040 instances for these classes), these will be designated *represented instances.*

When the system is being used by a child, the letter being trained is always pre-chosen by the child (or therapist), therefore, even before any input starts it is known to the system which letter is being trained. The writing variation however is unknown, in order to teach or 'learn' from the user this information must be obtained automatically after receiving the input. Using the subclasses defined in section 4.1, and the recognition techniques explained in section 3.4 the information can be collected with the result having an associated confidence as explained in the respective section. This operation

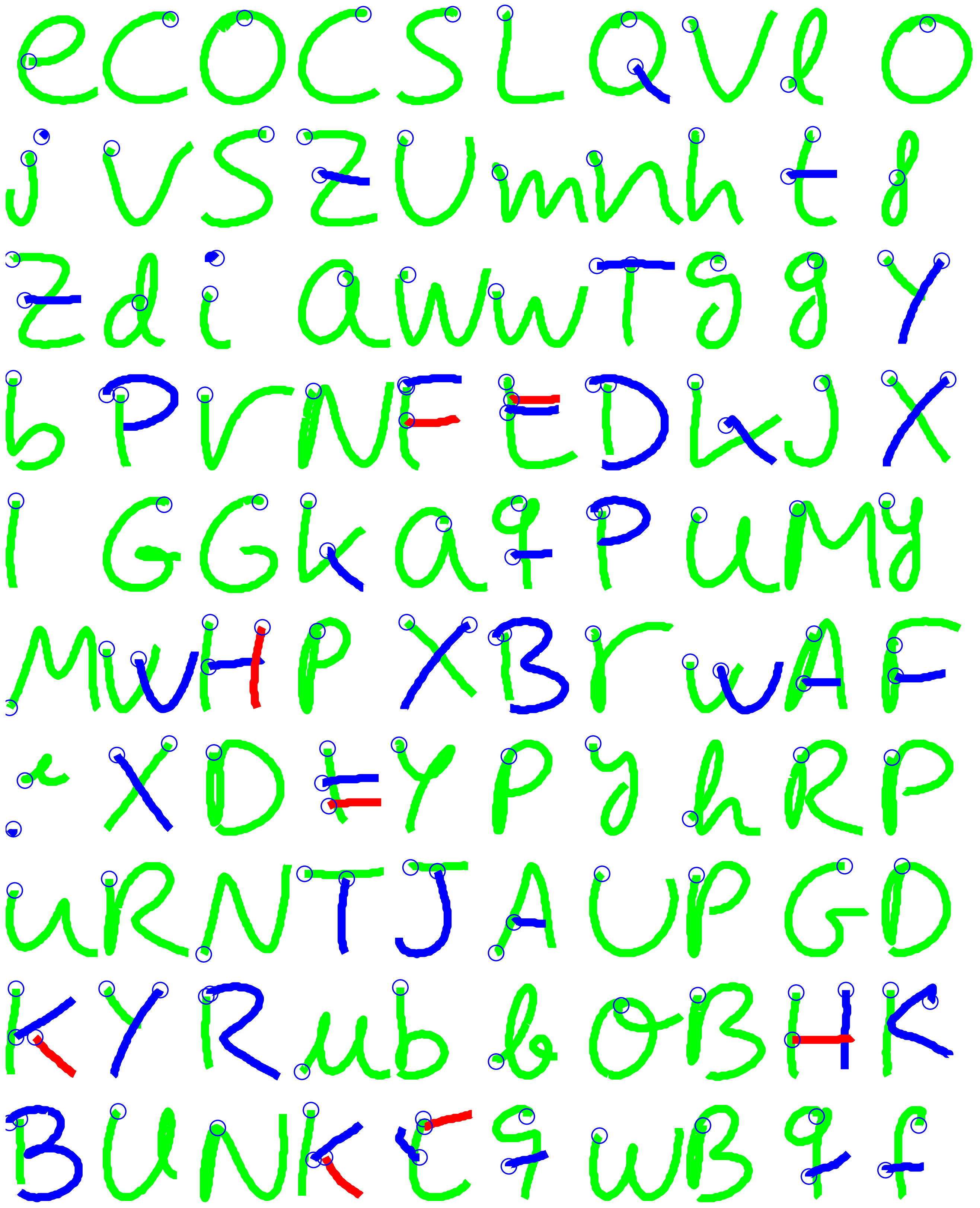


Figure 4.7: This figure shows the master characters for the biggest 100 subclasses, all lower and upper-case letters of the abecedarium are represented in this subset of the subclasses. A total of **4708 instances were used to produce these 100 shapes, this covers 75.45% of the dataset.** The $1^o$ stroke is in green, $2^o$ stroke in blue and $3^o$ stroke in red.

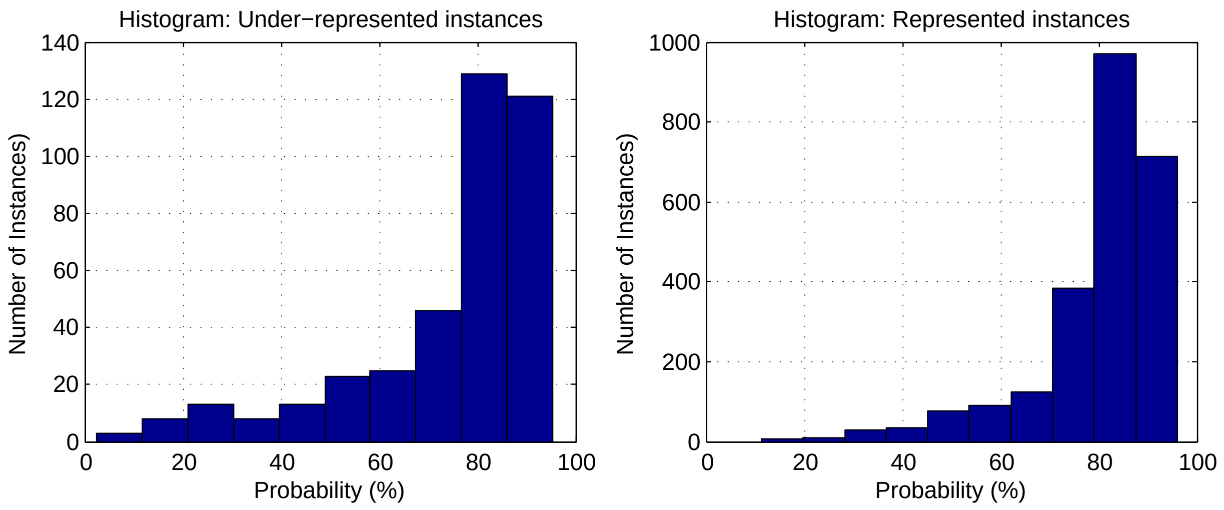


Figure 4.8: The first histogram shows a higher number of low confidence classifications (averaging 74%) when comparing with the second (averaging 80%).

was simulated in all the *represented and under-represented instances* of the testing dataset, . Figure 4.8 shows the results, it was achieved a relatively average high classification confidence (of 80% and 74% for *represented and under-represented instances* respectively), with an average classification time of $0.9943s$ per instance. From the histograms in figure 4.8, can be draw the conclusion that a higher success in classification is achieved when the training dataset size is increased in relation to the testing dataset. In practice this means that the larger the training dataset size, the more efficient the system will be.

## 4.3 System Output Production

As explained in section 3.5 the system is able to implements two learning paradigms with corresponding outputs. In both, it is necessary to adapt the system's shape output with the user's shape input as interaction progresses. The morphing technique developed easily handles this task, even allowing to define the speed of learning (i.e. how close the starting shape approximates the target shape) by tweaking the coefficients of the linear combination. Figure 4.9 shows an example of this procedure.

When operating in 'Learning by teaching' paradigm it is also necessary to generate characters with lower calligraphic correctness, the methodology defined in section 3.5.1 allows to achieve this, figures 4.10, 4.11, 4.12, 4.13, 4.14 and 4.15 show the effects of this operation in several master characters.

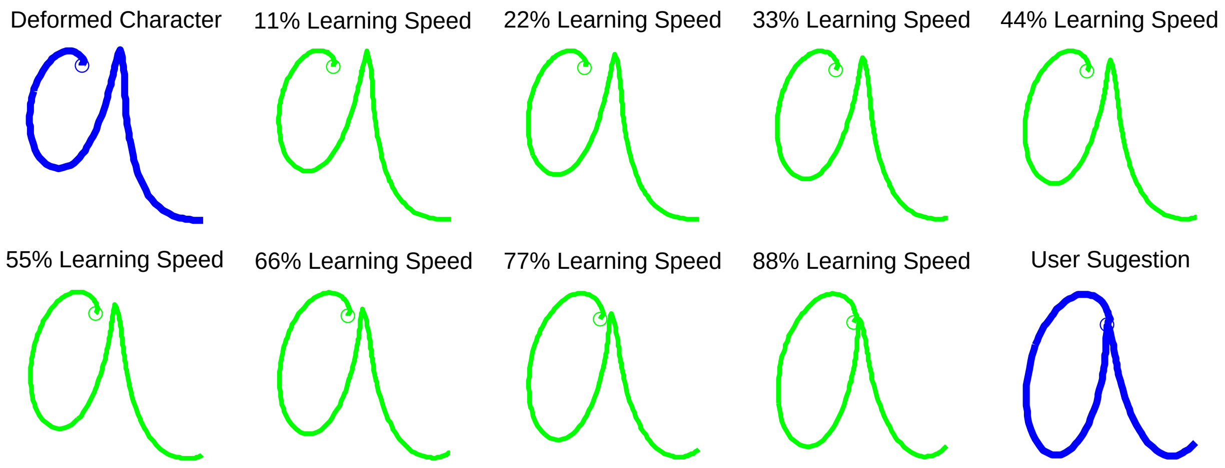


Figure 4.9: Example of an approximation of a character towards a user suggestion. The learning speed parameter allows to choose how fast the system 'learns' with the user suggestion.

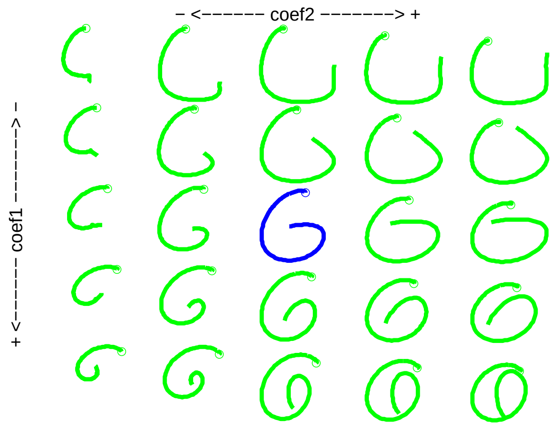


Figure 4.10: Effect of the deformation coefficients in 'G' character

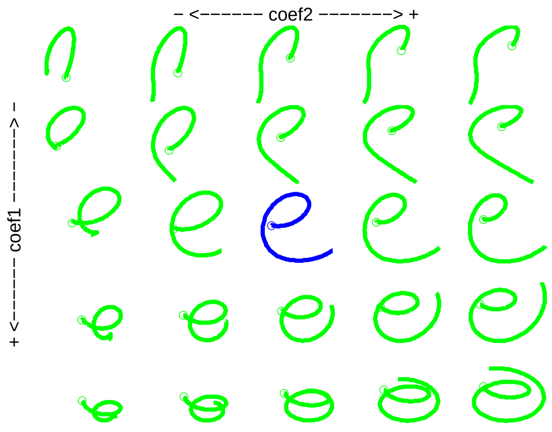


Figure 4.11: Effect of the deformation coefficients in 'e' character

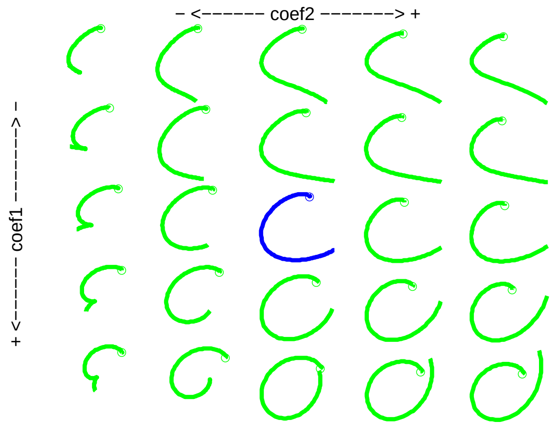


Figure 4.12: Effect of the deformation coefficients in 'C' character

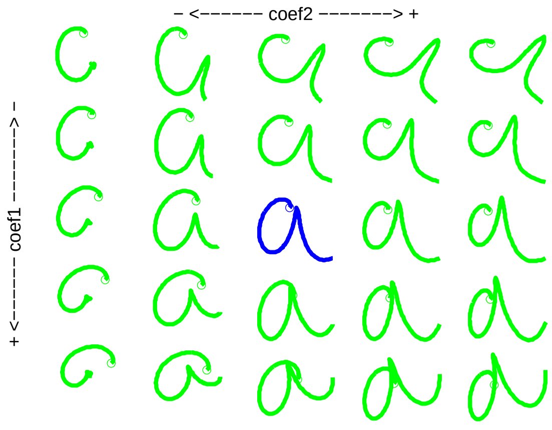


Figure 4.13: Effect of the deformation coefficients in 'a' character

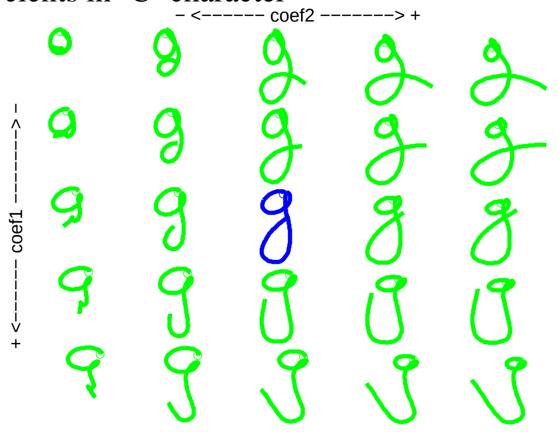


Figure 4.14: Effect of the deformation coefficients in 'g' character

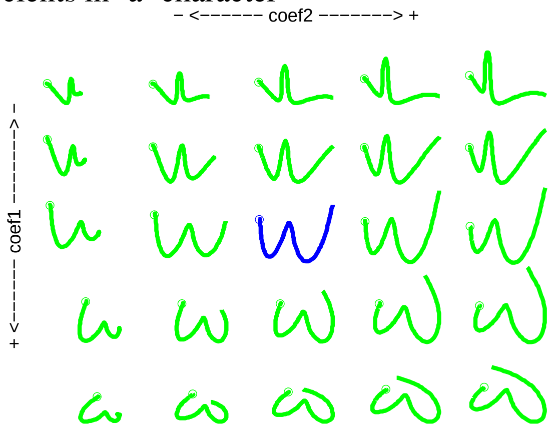


Figure 4.15: Effect of the deformation coefficients in 'w' character

# Chapter 5

# Conclusions and Future Work

The crescent inclusion of digital technologies in human society opens the path to promising new ways of solving various social problems. In the context of this work, the author believes a great contribution was made towards the creation and viability of an autonomous handwriting learning support system.

The framework developed fully meets all the objectives defined. It was possible to prove that from a base dataset, the framework is able to fully automatically generate higher and lower calligraphic quality characters on demand, and that the classification techniques can achieve a powerful, robust and flexible differentiation degree. Shape information is fully preserved and used in the classification, not only enabling to divide characters of different variations, but also allowing to divide characters of the same variation but with different drawing path. This capability allows the master characters to be displayed (not instantaneously but) as being written by a human intervener, enabling a very straightforward future integration with a humanoid robot.

The author reckons that the techniques created allow the future development of many interesting applications not only in the Handwriting Area, but in the context of Geographic Information Systems (GIS) and many others. An interesting handwriting application could be the studying of handwriting efficiency. Using a large enough dataset subclasses, every possible variation of every letter can be obtained. Using all the variations and a dataset of a language words along with respective recurrent frequency, can be estimated the most efficient way of writing every letter (in terms of pen tip path, writing hand ergonomics, etc.).

Future refinement work can be done to achieve a faster automatic dataset division by using a reduced warping window (envelope) for the DTW calculations, or executing a pre-matching of the features before proceeding to signature alignments. The use of multidimensional graphs and respective clustering algorithms to classify the dataset using signatures and chord-length principle might further increase the system differentiation ability. Finally, for characters with strokes composed of straight lines a the refinement of the *aaad* metric might further increase the system effectiveness in classifying shapes.

# Bibliography

[1] Roy F Baumeister. *Self-esteem: The puzzle of low self-regard.* Springer Science & Business Media, 2013.

[2] Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

[3] Carol A Christensen. The role of orthographic–motor integration in the production of creative and well-structured written text for students in secondary school. *Educational Psychology*, 25(5):441–453, 2005.

[4] Teófilo Emídio de Campos, Bodla Rakesh Babu, and Manik Varma. Character recognition in natural images. In *VISAPP (2)*, pages 273–280, 2009.

[5] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[6] Deanna Hood, Séverin Lemaignan, and Pierre Dillenbourg. When children teach a robot to write: An autonomous teachable humanoid which uses simulated handwriting. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 83–90. ACM, 2015.

[7] Monica MP Hoy, Mary Y Egan, and Katya P Feder. A systematic review of interventions to improve handwriting. *Canadian Journal of Occupational Therapy*, 78(1):13–25, 2011.

[8] Ragnheidur Karlsdottir and Thorarinn Stefansson. Problems in developing functional handwriting. *Perceptual and motor skills*, 94(2):623–662, 2002.

[9] Tomas Kulvicius, KeJun Ning, Minija Tamosiunaite, and Florentin Wörgötter. Modified dynamic movement primitives for joining movement sequences. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2275–2280. IEEE, 2011.

[10] Séverin Lemaignan, Alexis David Jacq, Deana Hood, Fernando Garcia, Ana Paiva, and Pierre Dillenbourg. Learning by teaching a robot: The case of handwriting. Technical report, 2016.

[11] David Llorens, Federico Prat, Andrés Marzal, Juan Miguel Vilar, María José Castro, Juan-Carlos Amengual, Sergio Barrachina, Antonio Castellanos, Salvador España Boquera, JA Gómez, et al. The ujipenchars database: a pen-based database of isolated handwritten characters. In *LREC*, 2008.

[12] Kathleen McHale and Sharon A Cermak. Fine motor activities in elementary school: Preliminary findings and provisional implications for children with fine motor problems. *American Journal of Occupational Therapy*, 46(10):898–903, 1992.

[13] Daigo Muramatsu and Takashi Matsumoto. An hmm online signature verifier incorporating signature trajectories. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 438–442. IEEE, 2003.

[14] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.

[15] Cynthia A Rohrbeck, Marika D Ginsburg-Block, John W Fantuzzo, and Traci R Miller. Peer-assisted learning interventions with elementary school students: A meta-analytic review. *Journal of Educational Psychology*, 95(2):240, 2003.

[16] Adrian D Sandler, Thomas E Watson, Marianna Footo, Melvin D Levine, William L Coleman, and Stephen R Hooper. Neurodevelopmental study of writing disorders in middle childhood. *Journal of Developmental & Behavioral Pediatrics*, 13(1):17–23, 1992.

[17] Rosemary Sassoon. *Handwriting: A new perspective*. Stanley Thornes, 1990.

[18] Ruben Tolosana, Ruben Vera-Rodriguez, Javier Ortega-Garcia, Fierrez, and Julian. Optimal feature selection and inter-operability compensation for on-line biometric signature authentication. In *2015 International Conference on Biometrics (ICB)*, pages 163–168. IEEE, 2015.

[19] Stijn Marinus Van Dongen. Graph clustering by flow simulation. 2001.

[20] Wenwu Yang and Jieqing Feng. 2d shape morphing via automatic feature matching and hierarchical interpolation. *Computers & Graphics*, 33(3):414–423, 2009.

[21] Hang Yin, Patrıcia Alves-Oliveira, Francisco S Melo, Aude Billard, and Ana Paiva. Synthesizing robotic handwriting motion by learning from human demonstrations.