Hindawi Publishing Corporation
The Scientific World Journal
Volume 2015, Article ID 323575, 18 pages
http://dx.doi.org/10.1155/2015/323575

Research Article

ASM Based Synthesis of Handwritten Arabic Text Pages

Laslo Dinges,[1] Ayoub Al-Hamadi,[1] Moftah Elzobi,[1] Sherif El-etriby,[2,3] and Ahmed Ghoneim[4,5]

[1]*Institute for Information Technology and Communications (IIKT), Otto-von-Guericke-University Magdeburg, 39016 Magdeburg, Germany*
[2]*Umm Al-Qura University, Makkah 21421, Saudi Arabia*
[3]*Faculty of Computers and Information, Menoufia University MUFIC, Menofia 32721, Egypt*
[4]*Department of Software Engineering, College of Computer Science and Information Sciences, King Saud University, Riyadh 11451, Saudi Arabia*
[5]*Department of Computer Science, College of Science, Menoufia University, Menofia 32721, Egypt*

Correspondence should be addressed to Laslo Dinges; laslo.dinges@ovgu.de

Received 7 January 2015; Revised 28 April 2015; Accepted 29 April 2015

Academic Editor: Tongxing Li

Document analysis tasks, as text recognition, word spotting, or segmentation, are highly dependent on comprehensive and suitable databases for training and validation. However their generation is expensive in sense of labor and time. As a matter of fact, there is a lack of such databases, which complicates research and development. This is especially true for the case of Arabic handwriting recognition, that involves different preprocessing, segmentation, and recognition methods, which have individual demands on samples and ground truth. To bypass this problem, we present an efficient system that automatically turns Arabic Unicode text into synthetic images of handwritten documents and detailed ground truth. Active Shape Models (ASMs) based on 28046 online samples were used for character synthesis and statistical properties were extracted from the IESK-arDB database to simulate baselines and word slant or skew. In the synthesis step ASM based representations are composed to words and text pages, smoothed by B-Spline interpolation and rendered considering writing speed and pen characteristics. Finally, we use the synthetic data to validate a segmentation method. An experimental comparison with the IESK-arDB database encourages to train and test document analysis related methods on synthetic samples, whenever no sufficient natural ground truthed data is available.

1. Introduction

A crucial step for every pattern recognition system is to train the classifier against a database and validate the system using the corresponding ground truth (GT). However, collecting handwriting samples is known as error prone, labor-, and time-expensive process [1]. Particularly costly are databases, which are suitable for training and validation of methods, that segment words into letters or analyse text pages as historical documents, since corresponding GT needs to include additional information. For real handwriting databases this has to be done in a time consuming manual or semimanual way. This is one of the reasons, why data synthesis recently gained more and more interest [2].

The problem of the lack of satisfactory handwriting databases is very obvious in case of Arabic handwritings, where there mainly two well known, free available (offline) word databases (IFN/ENIT) [3], which exclusively contains Tunisian town names, and the IESK-arDB [4] that contains international town names and common terms as well, as 280 historical manuscript pages and 6000 segmented characters. We developed the IESK-arDB word database as a general database to train and validate segmentation based recognition of Arabic words. Therefore, the writers are from different Arabic countries; however, all of them writing in standard Nask. To match even the requirements of explicit segmentation, we add detailed, manual GT for all word samples, which includes bounding boxes of Pieces of Arabic Words (PAWs)

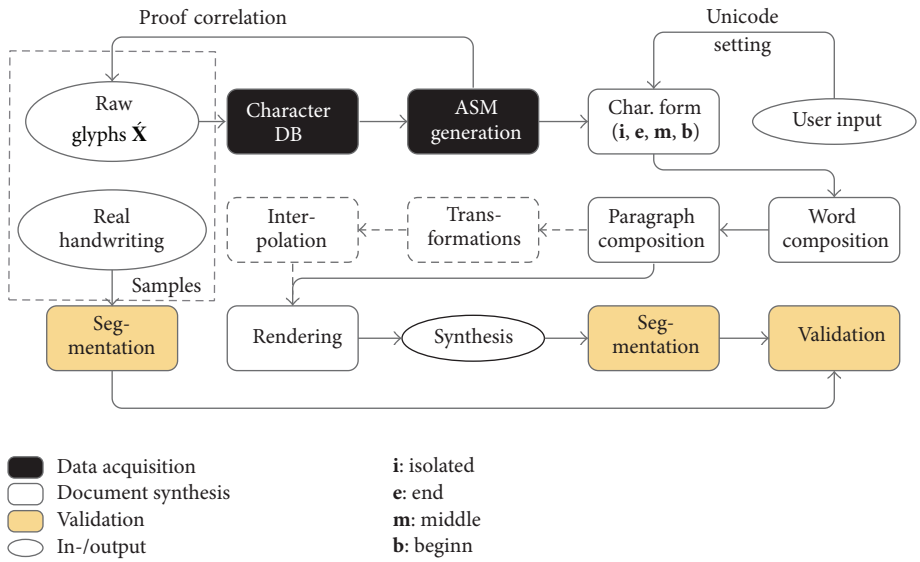


Figure 1: Simplified flowchart of the proposed synthesis system.

and points where two letters are connected. Nevertheless the IESK-arDB word database contains only 300 different words (written by 20 writers), due to the time expensive GT. The manual, proper generation of such detailed GT for complete Arabic text pages is indeed more complicated and hard to realize even for few samples. To bypass this problem, it would be very helpful, if databases necessary for the research field could be produced automatically. One possible way to accomplish this task is to generate synthetic handwriting samples of single words and texts.

We developed a system for this purpose, which allows creating synthetic databases from text files or Unicode that is entered within the user interface (UI). Figure 1 gives an overview of the design of that system. Ground truth are automatically generated. They contain original Unicode, ArabTeX transliteration, and further data as the bounding box of every letter. Furthermore the trajectories are stored for online applications. The system is capable of generating realistic synthesis of words, text lines, and complete (single column) text pages.

The rest of the paper is organized as follows. In Section 3 we give an overview of the related work. Thereafter we outline necessary data acquisition steps in Section 4. We also detail the mathematical background of Active Shape Models (ASMs) that we use to generate a large number of polygonal variations of Arabic letters. In Section 5 we describe our proposed methods to synthesize words and text pages by composing and arranging ASM based glyphs. Experimental results are discussed in Section 6, where we use synthesized databases to validate a segmentation method. Conclusion and future work are presented in the last section.

2. Arabic Script

The Arabic script has some special characteristics, so that synthesis or OCR approaches for Latin script will not succeed without major modifications [6]. Important aspects of Arabic script are as follows:

(1) Arabic is written from right to left.

(2) There are 28 *letters* (Characters) in the Arabic alphabet, whose shapes are sensitive to their form (isolated, begin, middle, and end); see Table 1.

(3) Six characters can only assume the isolated or end form, which splits a word into two or more parts, called *Piece of Arabic word* (PAW). They consist of the main body (connected component) and related diacritics (dots), supplements like Hamza (أ). In case of handwriting, the ascenders of the letters Kaf (ك), Taa (ط), or Dha (ظ) can also be written as fragments.

(4) Arabic is semicursive: within a PAW, letters are joined to each other, whether being handwritten or printed.

(5) Very often PAWs overlap each other, especially in handwritings.

(6) Sometimes one letter is written beneath its predecessor, like Lam-Ya (لى) or Lam-Mim (لم), or it almost vanishes away when it is in middle form, like Lam-Mim-Mim (لمّ) (unlike the middle letter of Kaf-Mim-Mim (كمّ)). Hence, in addition to the four basic forms, there are also special forms, which can be seen as exceptions. Additional, there are a few ligatures, which are two following letters that build a completely new character like LamAlif (لا).

Table 1: Arabic alphabet, Naskh style.

	i	e	m	b
Alif	ا	ـا		
Ba	ب	ـب	ـبـ	بـ
Ta	ت	ـت	ـتـ	تـ
Tha	ث	ـث	ـثـ	ثـ
Jim	ج	ـج	ـجـ	جـ
Ha	ح	ـح	ـحـ	حـ
Kha	خ	ـخ	ـخـ	خـ
Dal	د	ـد		
Thal	ذ	ـذ		
Ra	ر	ـر		
Zai	ز	ـز		
Sin	س	ـس	ـسـ	سـ
Shin	ش	ـش	ـشـ	شـ
Sad	ص	ـص	ـصـ	صـ
Dhad	ض	ـض	ـضـ	ضـ
Taa	ط	ـط	ـطـ	طـ
Dha	ظ	ـظ	ـظـ	ظـ
Ayn	ع	ـع	ـعـ	عـ
Ghayn	غ	ـغ	ـغـ	غـ
Fa	ف	ـف	ـفـ	فـ
Qaf	ق	ـق	ـقـ	قـ
Kaf	ك	ـك	ـكـ	كـ
Lam	ل	ـل	ـلـ	لـ
Mim	م	ـم	ـمـ	مـ
Nun	ن	ـن	ـنـ	نـ
He	ه	ـه	ـهـ	هـ
Waw	و	ـو		
Ya	ي	ـي	ـيـ	يـ

(7) Some letters like Tha (ث), Ya (ي), or Jim (ج) have one to three dots above, under or within their "body."

(8) Some letters like Ba (ب), Ta (ت), and Tha (ث) only differ because of these dots.

3. Related Work

There are several applications of text synthesis, such as word spotting, CAPTCHAs, character recognition improvement, calligraphy, and others [2]. Accordingly there are different approaches of synthesis. In the literature, research addressing the issue of synthetic text generation can be classified into two main categories, top-down and bottom-up approaches. Both can be either based on offline or online techniques, according to the available samples and applications. Top-down approaches are typically based on neuromuscular models, that simulate the writing process itself [7, 8]. Therefore, script trajectories are seen as a result of character key points (built by the human brain when learning how to write), the writing speed, character size, and inertia, which finally leads to the curvature of handwritings. These approaches are focused more on the physical aspects of writing than the actual handwriting sample outcome. One typical application is to investigate diseases as Parkinson or Alzheimer that influence handwriting abilities [9].

Bottom-up approaches on the contrary model the shape (and possibly texture) of handwritings itself. Hence, bottom-up approaches are preferred in context of text recognition task like segmentation or handwriting recognition. Bottom-up approaches can be further categorized into generation of new samples of the same level and concatenation to more complex outcomes, such as words, that are composed from characters or glyphs [10].

A common generation technique is data perturbation, which is performed by adding noise to online or offline samples. Samples might be complete units as text lines or words, but single characters or glyphs (as syllables, ligatures, or modified letters) are used mostly [11]. In case of letters or glyphs, noise is often achieved by degradation of offline or online samples or random displacement of trajectories; transformations as shearing, scaling, or rotation are favored for perturbing words or text lines. Another generation technique is sample fusion that blends two or a couple of samples to produce new hybrid ones [12, 13]. A better statistical relevance can be achieved using model based generation [14, 15]. This initially requires the creation of deformable models, which represent a class by a flexible shape. Deformable models are often based on statistical information that must be extracted from sufficient samples (usually on character level). Then unlimited new representations of the same model class can be generated. At the same time, deformable models are capable of generating more realistic variances than data perturbation, which closely depict the peculiarities of the letter class. Examples for deformable models are Active Shape Models (ASMs) and novel Active Shape Structural Models (ASSMs), which are used for generating variances of simple drawings and signatures [16]. ASMs are also applied for the classification of Chinese letters [17].

Since documents in Latin based script might be handprinted, concatenation of such handwriting samples to units of higher levels can be done without connecting the samples [10]. Proper simulation of cursive handwriting requires at least partially connection though. There are approaches that connect offline samples directly [18] and those who use polynomial [19], spline [20, 21], or probabilistic models [22]. Due to the semi cursive style, connecting is mandatory in case of Arabic script.

Systems using the described techniques to synthesize handwritings have been built for different scripts and purposes. Wang et al. [23] proposed a learning based approach to synthesize cursive handwriting by combining shape and

physical models. Thomas et al. [24] have proposed synthetic handwriting method, for generation of CAPTCHAs' (*completely automated public Turing test to tell computers and humans apart*) text lines. After segmentation, samples for each letter are generated using shape models. In the synthesis process, a delta log-normal function is used to compose smooth and natural cursive handwriting. Multiple approaches specific to Latin script that are based on polynomial merging functions and Bezier curves have been documented in [20]. Miyao and Maruyama [25] have proposed a method to improve offline Japanese Hiragana character classification using virtual examples synthesized from an online character database.

In contrast to word synthesizing, little research has been done concerning text line, paragraph, or document synthesis. If synthesis of multiple text lines is considered at all, it is typically modeled as horizontal baselines that may be influenced by noise [21], or each baseline line is defined by a rotation angle [10]. Varga et al. [26, 27] presented a method for handwritten English text line synthesizing; their methodology starts by composing static image of the text line by perturbing and chaining of character templates. Then text line is drawn using overlapping strokes and delta-lognormal velocity profiles, as stated by delta-lognormal theory. Chaudhuri and Kundu proposed a system to synthesize handwritten Bangla script pages [28]. For page layout simulation they compute Gaussian distributions from natural text pages to model different features, namely, left margin angle, line orientation, interline gap, and line undulation.

Due to the characteristics if Arabic script, which were discussed in Section 2, existing systems and methods can not be directly used for Arabic. Margner and Pechwitz [29] suggest an image based perturbation approach for the generation of synthetic printed Arabic words. They add global noise of different degrees, simulating degradation, as artifact, which are caused by repeated copying. As for the problem of automatic synthesis of offline handwritten Arabic text, to the best of our knowledge, Elarian et al. [3, 30] are the first who published research work addressing this problem so far. They propose a straightforward approach to compose arbitrary Arabic words. The approach starts by generating a finite set of letter images from two different writers, manually segmented from IFN/ENIT database, and then two kinds of simple features (width and direction feature) are extracted, so they can be used later as a metrics in the concatenation step. Saabni and El-Sana proposed a system to synthesize Pieces of Arabic Words (PAW) (without diacritics) [31]. They use digital tablets to acquire online samples, which are randomly composed to PAWs. Thereafter a subset of the produced synthetic data is selected by clustering techniques to get a compact database. In [32] we proposed a system to generate Arabic letter shapes by ASMs built from offline samples. Subsequently, we developed a system to render images of Arabic handwritten words, concatenating ASM based samples and using transformations on word level as optional, second generation step [33].

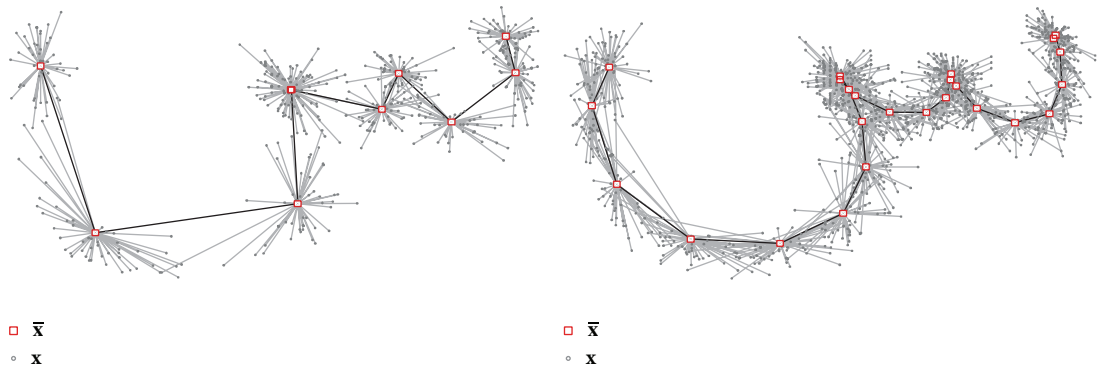
Table 2: Parameters and tresholds.

Parameter	Explanation	Value
r	Number of samples used to compute an ASM	50
m	Number of points of letter sample trajectory $\acute{\mathbf{x}}$ before approximation	$\overline{m} = 266.8$
n	Number of points of a approximated sample $\mathbf{x}$	25
ς	Controlling the space between words	user defined
α_r	Word rotation	$\pm 45^\circ$
α_s	Word slant	$\pm 45^\circ$
ϖ	Controlling the sharpness of synthesis contours	0–10
ω_p	Width of synthesized lines in points	1–10

Segmentation of Arabic words is quite challenging. It depends on holistic word as well as the features of single characters, and detailed GT are required to perform a useful validation. Hence, it is reasonable to test this method on synthetic databases, which contain such GT. One of the earliest segmentation based approaches, suggested for the recognition of Arabic handwritten text, is the one proposed by [34], and no segmentation results are reported though. Xiu et al. proposed probabilistic segmentation model, for this segmentation approach a tentative, contour based over segmentation is first performed on the text image; as a result, a set of what they called graphemes is produced [35]. The approach differentiates among three types of graphemes. The confidence of each character is calculated according to the probabilistic model, respecting other factors, for example, recognition output, geometric confidence, and logical constraint. The authors experimented the proposed methodology on five different test sets, achieving 59.2% success rate.

4. Data Acquisition and Generation

To synthesize Arabic handwritten words from glyphs, a sufficient amount of *glyph samples* has to be acquired first. In our case trajectories of single letters and their connections (Kashidas) are used as glyphs. The glyphs are acquired with online techniques since relevant information can be extracted more efficiently from trajectories than images. Nevertheless, we are mainly interested in synthesizing offline data. Hence, we decided to use online pens for data acquisition, for they can be applied as ordinary biros in contrast to digital tablets. Fifty or more samples per writer are taken from over hundred letter classes (28046 samples altogether) to built an online *character database*. To minimize manual effort and allow an easy extension, this database is completely automatically generated from raw data. Raw data are trajectories $\acute{\mathbf{X}} \in \mathbb{R}^{m\times 2}$ (see Table 2) for each stroke within a virtual DIN A4 page. That page has a resolution of 1000 dpi and 58 rps (reports per second), so there are constant timestamps $\triangle t = t_{i+1} - t_i = 0.0172$ s between neighbored $(\acute{x}_{i,1}, \acute{x}_{i,2}) \in \acute{\mathbf{X}}$. The generated database contains the trajectories and an image



(a) $n = 10$ (b) $n = 25$

FIGURE 2: The average $\bar{\mathbf{x}}$ and r samples $\mathbf{x}$ for the letter Sin in isolated form of writer 1. (a) The number of interpolation steps $n = 10$ is not sufficient to represent complex characters as Sin (س), and details will be lost or distorted. (b) With $n = 25$, the approximated samples $\mathbf{x}$ are reliable enough to build proper ASMs for all character classes.

representation for each Arabic letter class (see Table 1), as well as the resulting Active Shape Models (ASMs), which are described in the next section. Digits and special characters might be added in future work.

4.1. Computing Active Shape Models for Generation of Arabic Characters. Active Shape Models (ASMs) are statistical representations of the approximated shape of an object. An ASM uses the distribution of some significant points to store the most important information of many shapes of a class in one single model. Normally some well-defined landmarks have to be set manually for all samples and additional intermediate points between these landmarks are used if there are not enough landmarks to represent the shape. However, the definition of landmarks for over hundred classes for every writer is barely realizable and would prevent to add data from further writers efficiently. This is why we use two given landmarks, the *Start* and the *End* point of each polygon, and compute intermediate points by interpolating between the given $(\acute{x}_{i,1}, \acute{x}_{i,2})$ of $\acute{\mathbf{X}} \in \mathbb{R}^{m\times 2}$ to get polygons $\mathbf{X} \in \mathbb{R}^{n\times 2}$. By alternate storing of the $x_{i,1}$ and $x_{i,2}$ coordinates, each polygon is represented as a single vector of size $2n$: $\mathbf{x} = (x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \ldots, x_{n,1}, x_{n,2})^T$ that is required to build ASMs. Given the point numbers m of the original and n of the desired approximated samples, we interpolate in such a way that the time steps $\triangle_t = \acute{\triangle}_t m/n$ between neighbored interpolated points $\mathbf{p}_j, \mathbf{p}_{j+1} \mid \mathbf{p}_j = (x_{2j}, x_{2j+1})$ are still constant, the Euclidean distance however may vary. Given the original points $\acute{\mathbf{p}} \in \acute{\mathbf{X}}$ we compute an interpolated point

$$\mathbf{p}_j = (1-\lambda_a)\,\acute{\mathbf{p}}_{j'} + \lambda_a \acute{\mathbf{p}}_{j'+1} \quad \text{with } j' = \left\lfloor j\widetilde{\triangle_t} \right\rfloor,\ \widetilde{\triangle_t} = \frac{\acute{\triangle}_t}{\triangle_t},\ \lambda_a = j\widetilde{\triangle_t} - j', \tag{1}$$

where $j' \in \{1, m\}$ and $j \in \{1, n\}$. This enables a more detailed modeling of complex structures as the peaks of Sin (س) as one can see in Figures 2 and 3. We set $n = 25$, which is sufficient to represent even complex Arabic characters like Sin (س), as shown in Figure 2 (in order to speed up the synthesis process, n could be optimized for each individual class).

We then scale all samples $\mathbf{x}$ keeping their aspect ratio. Let W be the width and let H be the heights of an unscaled $\mathbf{x}$, and then we scale $\mathbf{x}$ by $100x100/(H * W)$ and translate $\mathbf{x}$ so that its center lies within the origin.

For each class there are r available vectors, from which the ASM of the corresponding class is calculated (we set $r = 50$ for our experiments).

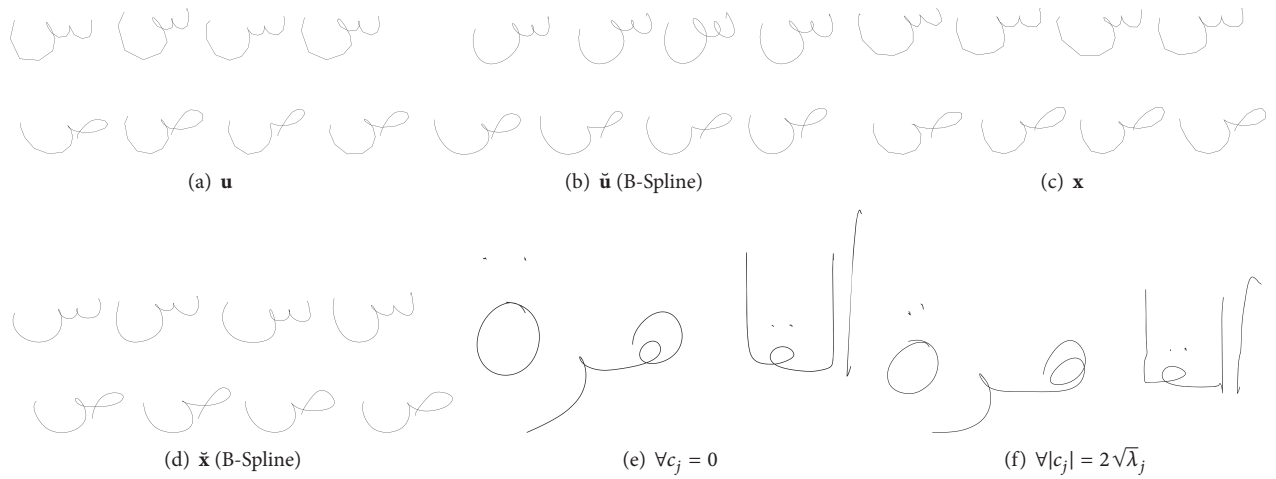
To build ASMs, the expected value $\bar{\mathbf{x}}$ and the covariance matrix $\mathbf{S}$ have to be calculated first:

$$\bar{\mathbf{x}} = \frac{\sum_{i=1}^{r} \mathbf{x}_i}{r}, \qquad \mathbf{S} = \frac{1}{2r-1}\sum_{i=1}^{r} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T. \tag{2}$$

As a consequence of the covariance matrix calculation, the Eigenvalues λ_i and Eigenvectors $\mathbf{e}_i$ can be determined, and then the ASM corresponding to character classes is calculated. Now any arbitrary number of vectors that represent each class can be calculated by linear combination:

$$\mathbf{u} = \bar{\mathbf{x}} + \sum_{j=1}^{r} c_j \mathbf{e}_j, \quad c_j \in \left[-2\sqrt{\lambda}_j, 2\sqrt{\lambda}_j\right]. \tag{3}$$

A limitation of c_j by $\pm 2\sqrt{\lambda}_j$ assures that all the deviations of $\mathbf{u}$ are within the doubled standard deviation σ. This is a common limit, since most training samples lie within this range. In few cases a limit of maximal $\pm 3\sqrt{\lambda}_j$ is applied, which



(a) $\mathbf{u}$ (b) $\breve{\mathbf{u}}$ (B-Spline) (c) $\mathbf{x}$

(d) $\breve{\mathbf{x}}$ (B-Spline) (e) $\forall c_j = 0$ (f) $\forall |c_j| = 2\sqrt{\lambda_j}$

Figure 3: Examples of glyphs: (a) examples for ASM-glyph representations $\mathbf{u}$ (where $\sum |c_j| \approx \sum 1.6\sqrt{\lambda_j}$; see (3)), (b) a B-Spline interpolation has been applied on the representations, (c), (d), original letter samples $\mathbf{x}$. A B-Spline interpolation with 5 steps has been applied on (b) and (e). Examples of ASM-glyphs $\mathbf{u}$ that are composed to words are shown in (e) and (f), where the influence is minimal (e) and maximal (f), respectively, for all eigenvalues.

requires a clear increase of r in order to keep $\mathbf{u}$ statistically reliable.

Some examples of $\mathbf{x}$ and $\mathbf{u}$ are shown in Figure 3. The computation of ASMs is quite costly, especially if many samples and interpolation points are used. This is why we avoid recalculations of ASMs at runtime, since the synthesizing process requires a minimum of 100 ASMs. Hence, all ASMs (including their corresponding online samples) are saved in files, to separate ASM generation from the synthesis module.

4.2. ASM Distance Measure. The distance δ of a sample $\mathbf{x}$ and a representation $\mathbf{u}$ of an ASM can be calculated as follows:

$$\delta(\mathbf{x}, \mathbf{u}) = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(u_{2i-1} - x_{2i-1})^2 + (u_{2i} - x_{2i})^2}. \quad (4)$$

Due to the performed scaling of $\mathbf{x}$, $\delta(\mathbf{x}, \mathbf{u})$ can be interpreted as the approximated deviation in percent. For all samples $\mathbf{x}$ we compute the most similar representation $\mathbf{u}^*$ by solving (3) numerically, using $\mathbf{c}$ as unknown (initially $\forall c_i = 0$). Some examples of $\mathbf{u}^*$ are visualized in Figure 4.

In contrast to ASMs from offline samples [32], most online based ASMs are capable of representing their input samples well, even without manually defined landmarks, since detailed, slowly written structures (where landmarks are suspected) are represented by more intermediate points. This effect allows an efficient extension and maintenance of the *character database*, since time consuming, manual landmarking can be avoided.

Nonetheless the average $\overline{\delta}$ is writer dependent. We measured $\overline{\delta} = 0.8$ for writer 1, $\overline{\delta} = 1.46$ for writer 2 and even higher values for the other writers, whose handwriting style is less tidy. Furthermore $\overline{\delta}$ depends on the letter class. A high $\overline{\delta}$ is mainly caused by some classes with diacritics as Zai (ز) that need improvement (see Figure 3(c)). The distance between the letters main body and diacritics can vary clearly, which may lead to inappropriate $\mathbf{u}^*$, in case that $\mathbf{x}$ is unlike $\overline{\mathbf{x}}$. To react against this effect, more training samples could be used or diacritics and main bodies could be separated using Active Shape Structural Models (ASSM) [16]. However, our ASMs are meant for synthesis approaches where the use of pure noise (perturbed data) is quite common, and hence moderate imprecision does not spoil the synthesis quality significantly.

5. Synthesizing Arabic Handwritings

In the following subsections we describe all methods that are applied to create synthetic Arabic handwritings from Unicode, using the *ASM data* from the last Section. Our system uses these either to show a few specific syntheses for preview purpose or to generate a complete database including ground truth.

The methods, which are used to synthesize and render Arabic handwritings, are described in the following Sections.

5.1. Composing Words. The basic idea of Arabic handwriting synthesis from Unicode is to select glyphs with proper shapes (isolated, initial, end, or middle form) and connect them subsequently to build PAWs, words, and texts, from which images or vector graphics are rendered.

5.1.1. Calculation of the Letter Form. Our system receives text as Unicode string that represents every letter as number. Since our samples are limited to the 28 regular letters and Tamarbuta (ة), we substitute special characters as Alif with Hamza above (أ) with their regular form Alif (ا) before starting the synthesis. Letter forms have a strong influence on the letter shape, but they are not given by regular Unicode; thus they have to be determined first.

(a) إس $\delta = 0.96$ (b) إي $\delta = 0.87$ (c) از $\delta = 5.6$

FIGURE 4: Examples of ASM based glyph $\mathbf{u}^*$ that has the highest correlation (deviation δ) with a randomly chosen samples $\mathbf{x}$. The average sample polygon $\bar{\mathbf{x}}$ is displayed to show how strong $\mathbf{x}$ differs from the expected shape. Some of the ASM for characters with diacritics are problematic, as shown in (c).

Letters of the set $\Gamma^2 = (\text{ادرزذو})$ can only assume **i**solated or **e**nd form. All other letters, such as Ayn() can assume begin-, middle-, and- as well as isolated- form. Ayn (عـ ـعـ ـع ع). Therefore, the form $\mathbf{f}_0$ of the first letter $\mathfrak{l}_0$ of a word is

$$\mathbf{f}_0 = \begin{cases} \mathbf{i} & \text{if } \mathfrak{l}_0 \in \Gamma^2 \\ \mathbf{b} & \text{else.} \end{cases} \tag{5}$$

Examples of the different letter forms within an Arabic word are shown in Figure 5. Let $\mathbf{f}_{-1}$ be the form of $\mathfrak{l}_{-1}$, which is the predecessor of letter $\mathfrak{l}$, and $\mathfrak{l}_{+1} = \varnothing$ define that the successor of $\mathfrak{l}$ is a space, tab, or return token $\varnothing$, and then the position $\mathbf{f}$ of a letter $\mathfrak{l}$ can be defined as follows:

$$\mathbf{f} = \begin{cases} \mathbf{i} & \text{if } \mathbf{f}_{-1} \in (\mathbf{i}, \mathbf{e}) \wedge \left(\mathfrak{l} \in \Gamma^2 \vee \mathfrak{l}_{+1} = \varnothing\right) \\ \mathbf{b} & \text{if } \mathbf{f}_{-1} \in (\mathbf{i}, \mathbf{e}) \wedge \left(\mathfrak{l} \notin \Gamma^2 \wedge \mathfrak{l}_{+1} \neq \varnothing\right) \\ \mathbf{m} & \text{if } \mathbf{f}_{-1} \in (\mathbf{b}, \mathbf{m}) \wedge \left(\mathfrak{l} \notin \Gamma^2 \wedge \mathfrak{l}_{+1} \neq \varnothing\right) \\ \mathbf{e} & \text{if } \mathbf{f}_{-1} \in (\mathbf{b}, \mathbf{m}) \wedge \left(\mathfrak{l} \in \Gamma^2 \vee \mathfrak{l}_{+1} = \varnothing\right). \end{cases} \tag{6}$$

Letters that have only two forms split an Arabic word into Pieces of Arabic Words (PAWs), which consist of one or more letters.

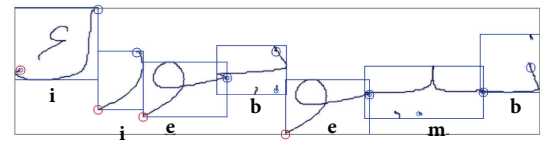


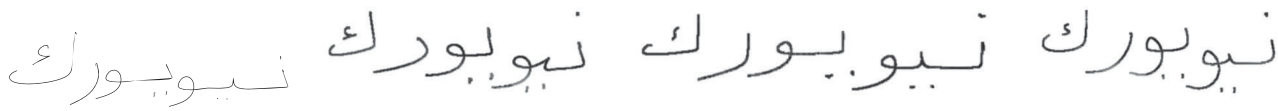
FIGURE 5: Outgoing from the Unicode sequence (ن ي و ي و ر ك) the forms (**i**solated, **e**nd, **m**iddle, and **b**egin) of all letters are determined (نـ ـيـ ـو ـيـ ـو ـر ك) and composed to the word نيويورك (New York).

5.1.2. Selection of Suitable Glyphs. To ensure that the styles of all neighbored glyphs are similar, glyphs of different writer are not mixed within a synthesis. We encouraged the writers to write letters only in this style that is dominant for their writings and avoid severe rotations. The size of the glyphs is normalized by the average character size we extracted from the IESK-arDB. However, we corrected the size manually in case of rare character classes. A suitability measure for the glyph joint points has not been considered, since steady joints are achieved by B-Spline interpolation and rendering at the end of the synthesis process.

5.1.3. Connecting Glyphs. After all letter classes are defined by their names and forms, corresponding ASMs are loaded.

Input: Current PAW $\mathbf{A}_c$, precedent PAW $\mathbf{A}_p$, average letter width $\bar{\mathbf{l}}_w$
Output: Shifted current PAW $\mathbf{A}_c$
If *Bounding boxes* of $\mathbf{A}_c$ *and* $\mathbf{A}_p$ *overlap* **then**
  while *Trajectories* of $\mathbf{A}_c$ *and* $\mathbf{A}_p$ *intersect* **do**
    Translate the x coordinates of $\mathbf{A}_c$ by $(1/4)\bar{\mathbf{l}}_w$;

Algorithm 1: Solving intersections of neighbored PAWs.



(a) Writer 1 (b) Writer 2 (c) Writer 3 (d) Writer 4

Figure 6: The word نيويورك (New York), synthesized by the average letter shapes $\overline{x}$ of different writers.

The ASMs are used to generate an unique polygonal representation $\mathbf{u}$ for each occurrence of a letter class in order to avoid piecewise identical syntheses. In order to compose words from these polygons, each letter in end or middle form has to be connected with its predecessor: عـر → عـر, نـ → نيـ → نيو. Let $\mathbf{p_f}$ be the first point of نيـ and $\mathbf{p_l}$ the last point of its predecessor نـ, and then we can connect them by translating نيـ by $\mathbf{p_l} - \mathbf{p_f}$. An example is shown in Figure 5.

The relation of a PAW's y coordinate and the baseline depends on the letter classes the PAW is composed of. Thus we extracted the average μ_r and variance σ_r of the relative distance between the baseline and the center of a letter from manual created ground truth of our (real) word database IESK-arDB [4]. We set the baseline to 0 and shift all y coordinates of each PAW by $\mathcal{N}(\mu_r, \sigma_r)$. Finally, the space between the rightmost point of a PAW and the leftmost point of its predecessor has to be defined. Therefore, the user depending parameter ς is used that may be negative in order to simulate overlapping PAWs. Given $\mathbf{p_l}$ and $\mathbf{p_f}$, the PAW can be translated by the vector

$$\mathbf{t}_{\text{paw}} = \begin{pmatrix} p_{l_1} - p_{f_1} - \varsigma \\ \mathcal{N}(\mu_r, \sigma_r) \end{pmatrix}. \quad (7)$$

In case of intersections between the polygons of overlapping PAWs, ς is increased iteratively by 25% of the average letter width, as described in Algorithm 1.

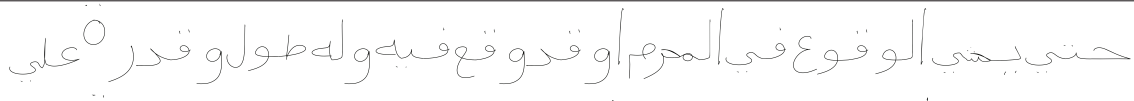
Examples of words composed by the average letter shapes of different writers are shown in Figure 6. Examples from ASM based and original samples can be found in Table 3. The maximal deviation σ_A means that the influence of an eigenvector $\mathbf{e}_i$ is limited by $\pm 2\sqrt{\lambda_i}$. While letters look similar using σ between 0 and 1, ASM representations $\mathbf{u}$ with σ_A of 2 already provoke increased letter variation. ASMs (trained with $r = 50$ samples) are barely capable of representing a deviation of $\sigma_A = 3$ though. As a matter of fact, there are noise based deformations as shown in Table 3 (second last row). This effect might be intended to create especially challenging syntheses.

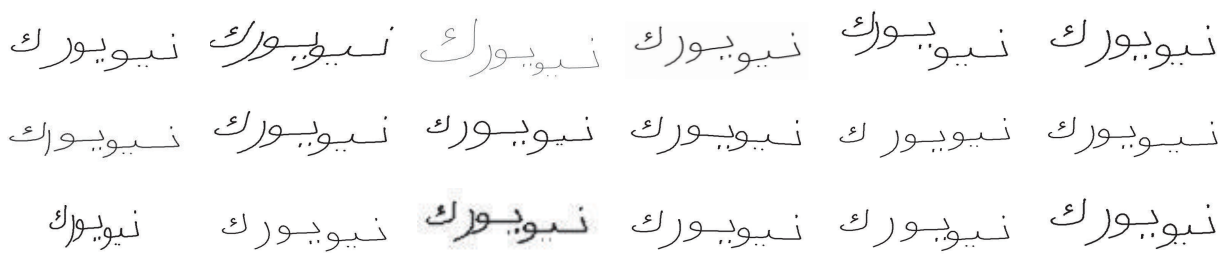
5.2. Simulation of Global Variances. ASMs already contain variations in slant, width or connection size. Nevertheless, these variations are limited by the used samples. In order to increase and control these variations, affine transformations are used (scaling, translation, shearing, and rotation), which allow optional manipulations of letter and PAW shapes. The user interface (UI) of our synthesis system allows to set the average μ and variance σ of a Gaussian distribution for all affine transformations. Particularly global variations as the slant can be achieved this way. The influences of these affine transformations on the resulting word image is shown in Figure 7.

A stretching is performed by scale each x component of each letter point by the factor $c \in [0.5, 2]$. The word slant can be set by shearing the word with an angle of α_s as the skew of PAW can be manipulated by rotating to the angle α_r, where $\alpha_s, \alpha_r \in [-45^\circ, +45^\circ]$. We analyzed the skew and slant of samples of the IESK-arDB database [4] with local minima regression and Hough transform. We found that the skew correlates with a Gaussian distribution of $\mathcal{N}(-3.8^\circ, 7.1^\circ)$ (passed Chi square test with $\alpha = 0.05$). The slant can be represented by $\mathcal{N}(-4.6^\circ, 14.4^\circ)$. The size of the complete word or single letters can be adjusted by an equal scaling with $c \in [0.1, 10]$, which can be used to control the resolution of the synthesized word images or to increase variation of letter size. As described in Section 5.1.3 variations of PAW positions can be realized by translations. Even using the same glyphs, synthetic words can assume large variations in shape when using affine transformations or cutting connection size, as shown in Figure 7.

When examining the glyph samples and synthesis we found that compared to complete handwritings, letter connections of the acquired samples are extended and even excessively (approximately twice as) long in case of writer 1. Hence, we allow to delete up to 25% of the letter points to simulate stretched or missed connections, which often occur

Table 3: Syntheses using ASMs and sample based glyphs of writer 1.

$\sigma_A \sim \overline{\lvert c_j \rvert}\sqrt{\lambda_i}$	Result
0	
1	
2	
3	
Samples	



(a) Stretching (b) Slant (c) Size (d) Skew (e) Position (f) Connection

Figure 7: Affine transformations and connection shortening on synthetic words.

in natural Arabic handwritings. In Arabic handwritings some characters as Ya (ي) are sometimes written beneath their predecessors. As a result they resemble a single character, which impedes segmentation and recognition tasks. This effect can be simulated by a strong reduction of the Kashida, as shown in Figure 8. However, this feature has been not yet entirely implemented, since a list of all pairs of letters that typically show this behaviour needs to be created first.

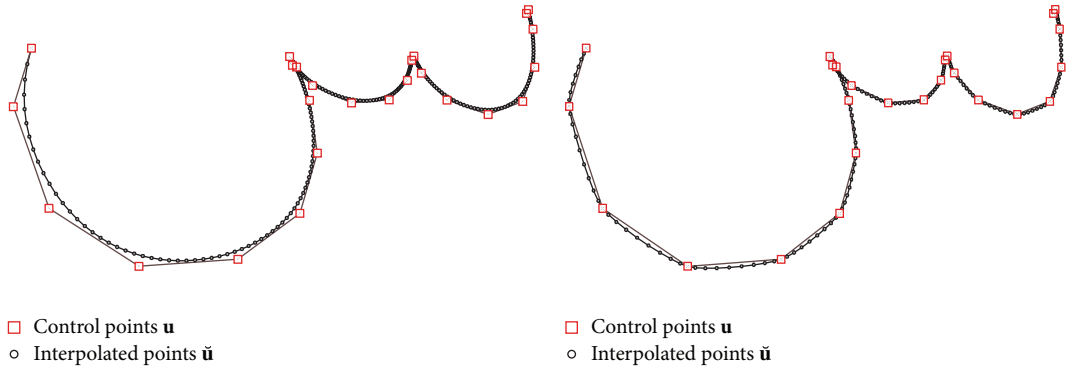
5.3. Interpolation. Since a polygonal letter representation does not look natural for vector graphics or images with a scaling factor > 1, we use interpolation to improve the outcome. Let $n(\mathbf{u})$ be the number of points of $\mathbf{u}$, and then we use $\mathbf{u}$ as control points to interpolate a curve $\breve{\mathbf{u}}$. By increasing the interpolation steps, $n(\breve{\mathbf{u}})$ can be approximated to the original number of points of a sample $\acute{\mathbf{X}}$. The average length of $\acute{\mathbf{X}}$ is $\overline{n}(\acute{\mathbf{X}}) = 266.8$, and hence a tenfold increase $n(\mathbf{u}) = 25 \rightarrow n(\breve{\mathbf{u}}) = 250$ is generally sufficient to achieve smooth syntheses. To ensure that the above mentioned methods work efficiently on the compressed representation $\mathbf{u}$, the interpolation step is applied just before rendering or skipped in case of low-resolution synthesis.

Our system supports two interpolation methods. Piecewise Cubic hermite-interpolation is C^1-steady, which means that only the first derivation of the used interpolation function is steady. Therefore, hermite-interpolation leads to less smooth and accurate results, a property, that can be used to create noisy handwritings [32] (perturbed data technique). State-of-the-art B-Spline interpolation is commonly used within CAD-applications, since it is C^2-steady and leads to smooth, natural curves, which can be defined properly by their control points, as shown in Figure 9. Although the B-Spline curve do not pass through all control points, it fits the original curve sufficiently when $n = 25$ control points are used. Given the measure δ of (4), the average distance between $\mathbf{u}$ and the n closest points of $\breve{\mathbf{u}}$ can be computed. Using a cross-validation with $k\text{fold} = 10$, we get $\overline{\delta}(\mathbf{u}, \breve{\mathbf{u}}) = 0.45$ with a variance of 0.058.

5.4. Rendering Handwriting Images. The former sections discussed how to compose and interpolate polygonal representations of Arabic handwritings. Now those have to be transformed into images and saved in common files formats (.bmp, .png, etc.), which can easily be loaded by most text

(a) (b) (c)

Figure 8: Example for a synthesized letter pair using (a) full (b) half and (c) negative *Kashida* length.



(a) B-Spline (b) Cubic

Figure 9: Comparison of B-Spline and Cubic interpolation where $n(\mathbf{u}) = 25$ and $n(\breve{\mathbf{u}}) = 250$.

recognitions or document analysis systems. Even preprocessing steps, as thinning, can influence the performance of the following tasks. Such preprocessing is sensitive to secondary features or flaws, which are a result of the used writing materials; hence the synthesized files should contain those features too. Therefore, we propose a rendering technique that reflects optical features caused by common pens as ball pens or pencils [33]. Subsequently, a modification of this technique is described that allows rendering features of historical handwritings.

First of all, pixels $\mathbf{o}$ have to be found that are close to polygons $\breve{\mathbf{u}}$ and belong to the foreground. As shown in Figure 10(a), we first interpolate between two neighbored points $\mathbf{p}, \mathbf{q}$ of $\breve{\mathbf{u}}$ (or $\mathbf{u}$) and get $\rho * \|\mathbf{p} - \mathbf{q}\|$ new points

$$\mathbf{o}' = \nu\mathbf{p} + (1 - \nu)\,\mathbf{q}, \quad \nu \in [0, 1], \tag{8}$$

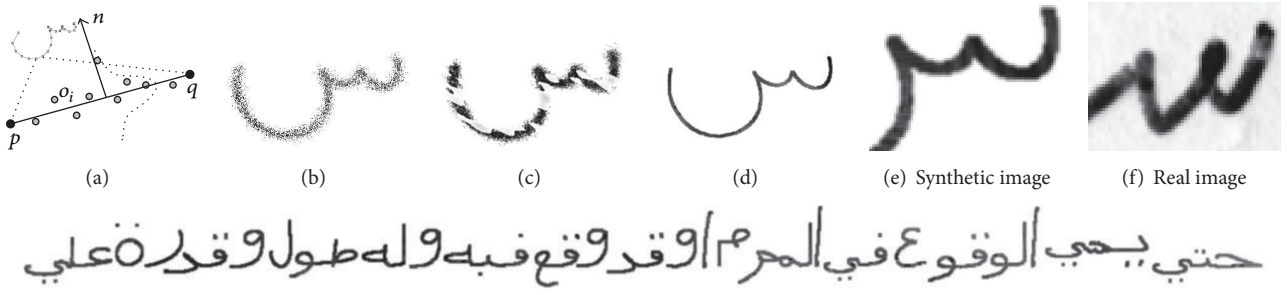
where ν is uniform distributed and ρ is a user defined parameter.

Let $\vec{\mathbf{n}}$ be the normal vector of line $\overline{\mathbf{pq}}$ and then we shift each $\mathbf{o}'$ along $\vec{\mathbf{n}}$ using a Gaussian distribution

$$\mathbf{o} = \mathbf{o}' + \mathcal{N}(\mu, \sigma)\,\vec{\mathbf{n}}, \tag{9}$$

where σ and μ define the line width, which is decreased up to 20%, in case that $\mathbf{p}$ lies between the *Pen Up* or *Pen Down* point and the neighbored control point. As a result, the prototype of a word image has been prepared that defines all pixels, which are influenced by pigments at all (shown in Figure 10(b)). To allow sharp contours, a limitation of $|\mathcal{N}(\mu, \sigma)| < \bar{\omega}\sigma$ is applied according to the simulated pen, where $\bar{\omega}$ is user dependent.

5.4.1. Texture. Biros or pencils cause an irregular pigmentation intensity, which is reflected by pixel intensities $I(x, y)$ of scanned images. A realistic physical model that simulates this behavior in a proper way is beyond the scope of this paper; a more simplified and generalized approach however is quite practical. Inspired by the ability of Fourier transform based image compression to represent the main nature of a texture by a small subset of the underlying frequencies, we define a texture by 10 points (λ, θ, ϕ) in frequency space in order to emulate irregularities caused by pen and paper. This way, simplified but unique textures can be created at runtime. However, in contrast to image compression, we are not interested in avoiding noise-prone high frequencies.



(a) (b) (c) (d) (e) Synthetic image (f) Real image

(g) Example text line

Figure 10: Painting technique: (a) scheme, (b) word shape with $\rho = 6$, $\sigma = 3.5$ (Gaussian filter is disabled), (c-d) example results for painting technique: (c) test texture applied on (b), (d) $\rho = 30$, $\sigma = 1.0$ with ball pen texture, and (e-f) comparison of natural and synthetic ball pen textures.

Hence, different high and low frequencies are combined to simulate regional as well as locale effects (behavior of ink, paper texture). Each point represents a texture layer in image space with $I_1(x, y) \in [-1, 1]$. We created several texture classes by defining different sets of Gaussian distributions for the angle θ, wave length λ, and phase shift ϕ of all layers manually. Afterwards, pixel intensities $I_2(\mathbf{o}) \in [0, 256]$ are computed by accumulating most texture layers. The left layers are used to achieve variations in their intensities by multiplication. Finally, an image $\mathbf{I}$ with a small margin is created and all $\mathbf{o}$ are fitted to $\mathbf{I}$ subsequently.

By creating word syntheses using polygonal glyphs and rendering techniques, smooth letter connections can be achieved more efficiently compared to synthesis approaches that use image based glyphs. Furthermore, in contrast to the usage of natural textures, the described technique is able to generate unique, nontiled textures for every synthesized word, as shown in Figures 10(c)–10(e). Depending on the texture class, a median or Gaussian filter is used before saving the image.

5.4.2. Simulation of Feather Like Writing Tools. The previous method allows a proper simulation of text that is written by ball pens, pencils, or coal on white paper. However, to allow a more accurate simulation of writing instruments as fountains pens or feathers, we extended our rendering technique. This includes mainly the implementation of two features: the writing speed and the shape of the top of the writing instrument, further called *Pen Shape*.

Pen Shape. If the *Pen Shape* is modeled as line or ellipsoid, the line width of the trajectory $\breve{\mathbf{u}}$ depends not only on the width of the *Pen Shape* w_P, but also on its angle γ_P and the angle of the trajectory tangent $\gamma_{\breve{\mathbf{u}}}$. We initialize $\gamma_P = 45^\circ$, changing it continuously with a maximal deviation of $\pm 15^\circ$. This locale deviation is computed by adding two cosine functions, where the wavelengths, phases, and amplitudes are redefined by Gaussian distributions for each synthesis.

According to the texture of a *Pen Shape*, the contact with the paper and consequently the caused pigmentation can vary. This is simulated by an one-dimensional function $f(x_P) \mid 0 \le x_P \le w_P$ that defines the pigmentation potential for the long axis of the *Pen Shape*. An emphasized example, inspired by a fountain pen, is shown in Figure 11(d).

Binary images can be rendered in a faster way by defining polygons (here a triangle mesh) that is a result of extruding an one-dimensional *Pen Shape* along $\mathbf{u}$ or $\breve{\mathbf{u}}$. A visualization of this can be found in Figure 11(c), and a resulting synthesis is shown in Figure 11(h). These polygons can then be drawn by standard routines. If the angle of the Pen Shape and the line is identical or so close, where the polygon width would be smaller than one pixel, a line have to be drawn instead to avoid letter fragmentation.

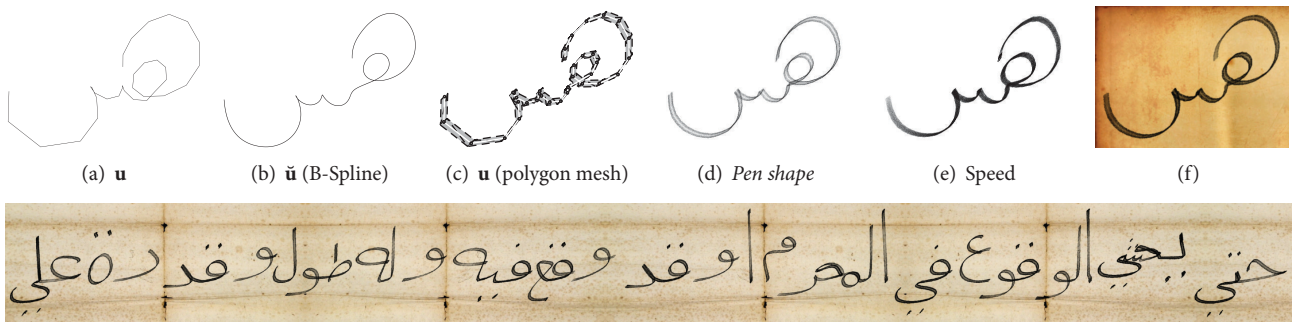
Writing Speed. Large lines or bows, as the left part of Sin (س), are usually written faster than more complex structures. A high writing speed often causes a lack of pigmentation that leads to brighter or dappled lines. However, there is no need to reconstruct writing speed, for the used online letter samples already that contain such information. From each representation $\breve{\mathbf{u}}$ we can extract the relative, local writing speed for ASMs representations in points per second

$$v(i) \text{ pt/s} = \frac{\left\| \left(\breve{u}_{2i-1}, \breve{u}_{2i}\right) - \left(\breve{u}_{2i+1}, \breve{u}_{2i+2}\right) \right\| \text{ pt}}{\left(n(\mathbf{u})/n(\breve{\mathbf{u}})\right) \Delta_t \text{ s}}, \quad 1 \le i \le n(\breve{\mathbf{u}}). \tag{10}$$

In order to estimate the expected pigmentation intensity, we use the normalized writing speed $\dot{v} \in [0, 1]$, where $\dot{v} = 0$ is the slowest and $\dot{v} = 1$ is the fastest part of the trajectory of a synthesis. If $\mathfrak{c}_0$ is the current color of a pixel in RGB color space and $\mathfrak{c}_p$ is the color of the pen pigment, then the new color is computed by

$$\mathfrak{c} = \mathfrak{c}_p \alpha_a + \mathfrak{c}_0 \alpha_b \left(1 - \alpha_a\right), \quad \mathfrak{c} \in \mathbb{R}^3, \tag{11}$$

where $\alpha_a = 0.8(1 - \dot{v}) + 0.2$ defines opacity of $\mathfrak{c}_p$ and $\alpha_b = 1$ defines the opacity of the background. Although pigments that are used for handwritings are typically full or semiopaque, reduction of pigmentation caused by increased writing speed can lead to transparency. This is simulated by reducing α_a, as shown in Figure 11(e).



(a) **u** (b) $\breve{\mathbf{u}}$ (B-Spline) (c) **u** (polygon mesh) (d) *Pen shape* (e) Speed (f)

(g) Example of a synthesized text line

(h) Example of a synthesized text line (fast method for binary images)

FIGURE 11: From the polygon **u** to a synthetic image that simulates a feather like writing instrument. (c) Polygon mesh created from **u** that is used for fast rendering of binary images.

To ensure a steady behavior, we interpolate the speed of connected letters. For the first $n(\breve{\mathbf{u}})/4$ points $(\breve{u}_{2i-1}, \breve{u}_{2i})$ we set the speed $\dot{v}$ of a letter in middle or end form to

$$\dot{v}(i) := \dot{v}(i)\left[1-\lambda_i\right] + \dot{v}_b\left(n(\breve{\mathbf{u}})\right)\lambda_i,$$
$$\lambda_i = 4\frac{i}{n(\breve{\mathbf{u}})},\ i \le \frac{1}{4}n(\breve{\mathbf{u}}), \tag{12}$$

where $\dot{v}_b(n(\breve{\mathbf{u}}))$ is the speed at the end of the previous letter. The effect on the synthesis is shown in Figure 11(d).

Render on Degraded Background. Finally, we create Figure 11(f) by combining our texture rendering technique of Section 5.4.1 with the one based on *Pen Shape* and writing speed, using a nonuniform background. Therefore, we applied a transparent texture on all pixels that does not belong to the background texture. This way, small, random irregularity is simulated.

In the shown examples a black color is used, as most documents are written with dark ink (like iron oxide or soot). Pigmentation intensity is implemented as transparency, which allows simulating pigment accumulation at crossing lines or in case of a textured background. However, also colored opaque or semiopaque ink can be simulated. This might be interesting in the context of historical documents, since important passages are often highlighted by using red ink. Another important feature of historical documents is degradation of paper or parchment. Currently we simply use images of natural paper or parchment as background textures, which are scaled or tiled in case where they are smaller than the synthesized document.

5.5. Generation of Text Pages. Recently, not only character or word recognition but also more complex document analysis issues that address the interpretation or recognition of complete documents become a focus of Arabic handwriting research. Hence, we investigate the possibilities of text page synthesis. In this regard the accurate simulation of the lower baselines is crucial. Many problems that occur while detecting lines, words, or connected components depend on these baselines, as for instance assigning diacritics (that are very close to more than one text lines) to their corresponding PAW.

Our approach of simulating baselines has three steps. First of all we set the coordinates (x_0, y_0) of the first letter for each line. Secondly the curvatures of the baselines have to be computed. Thirdly potential intersection of words have to be solved.

Start PAWs. Due to the style of Arabic handwritings, all lines start at the rightmost x-coordinate $x_0 = x_{\max}$. The vertical space between two neighbored lines can be defined as percentage of the average PAW heights to get y_0 for each baseline.

Baselines. We implement the second step by declaring functions $f(x, y_0)$ that define the curvatures of the simulated Baselines Ξ_s (y_0 defines the initial heights of a line). We normalize $\acute{y}_0, \acute{x} \in [0,1]$ inside $f(x, y_0)$, so it becomes independent of the page size. How to compute proper f_i is explained in Section 5.6.

Let $\mathbf{l}^r$ define the lower right of a letters bounding box, then we first translate the letter $\mathbf{l}$ towards the baseline, so that $\mathbf{l}^r$ touches it. Subsequently, the y-position of $\mathbf{l}$ must be corrected according to its class by translating it by $\mathbf{t}_b$, so that $\mathbf{l}^r$ might be above or below $f(l^r_x, y_0)$, as shown in Figure 12. Therefore, we extracted the normalized statistical relation $\mathcal{N}_b$ between $\mathbf{l}^r$ and the baseline from the IESK-arDB for all letter classes. If l^h is the heights and $\mathbf{l}^i$ is the ith point of $\mathbf{l}$, we can compute $\mathbf{t}_b = (0, l^h \mathcal{N}_b)^T$ and $\mathbf{l}'^i = \mathbf{l}^i + \mathbf{t}_b$ as shown in Figure 12. We perform this for every first letter $\mathbf{l}_f$ of a PAW. Apart from the position,

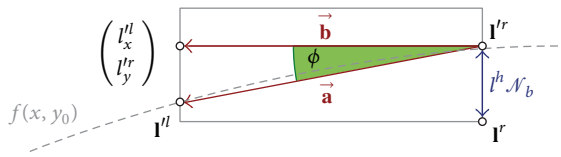


FIGURE 12: Arranging a letter along the baseline.

also the PAW skew depends on the baseline. We simulate this by rotating all points $\mathbf{l}'^i$ of a letter around their *pen down* point $\mathbf{l}'^1$. Let $\mathbf{l}^l$ be the lower left of the bounding box of the current letter $\mathbf{l}$, and then the rotation angle is

$$\phi = \cos^{-1} \frac{\langle \vec{\mathbf{a}}, \vec{\mathbf{b}} \rangle}{|\vec{\mathbf{a}}| \left|\vec{\mathbf{b}}\right|}, \quad \vec{\mathbf{a}} = \mathbf{l}'^r - \mathbf{l}'^l, \ \vec{\mathbf{b}} = \mathbf{l}'^r - \begin{pmatrix} l_x'^l \\ l_y'^r \end{pmatrix}. \tag{13}$$

Before we rotate a letter $\mathbf{l}' \neq \mathbf{l}'_f$ it has to be connected with its predecessor $\mathbf{l}'_p$ translating it by $\mathbf{t}_j = (\mathbf{l}'^e_p - \mathbf{l}'^1)$. This way, the handwriting synthesis fits to the baseline without causing aliasing effects.

Solving Intersections. In the last step, we detect and solve intersections between lines. Therefore, all PAW $\mathfrak{B}$ of the line above have to be detected, whose bounding boxes overlap with the current PAW $\mathfrak{A}$ or whose distances are less than ϵ. For all $\mathfrak{B}$ we then calculate weather there is any intersections between line segments $\mathfrak{s}_\mathfrak{A} = \overline{\mathbf{l}'^i_\mathfrak{A} \mathbf{l}'^{i+1}_\mathfrak{A}} \in \mathfrak{A}$ and $\mathfrak{s}_\mathfrak{B} = \overline{\mathbf{l}'^i_\mathfrak{B} \mathbf{l}'^{i+1}}_\mathfrak{B} \in \mathfrak{B}$. If so, $\mathfrak{A}$ is translated by $(0, \psi)^T$. This has to be done also for the predecessor of $\mathfrak{A}$, by using the translation $(\psi, 0)^T$. Similar to Algorithm 1 (where intersections of two neighbored PAWs of the same line are handled) these steps have to be repeated, until no intersection is detected anymore.

Two lines, which are not parallel, have an intersection point $\mathbf{z}$. To proof weather two line segments intersect, $\mathbf{z}$ of their corresponding lines must be calculated first. A pair of line segments $\mathfrak{s}_\mathfrak{A}$ and $\mathfrak{s}_\mathfrak{B}$ intersects, if and only if $\mathbf{z}$ lies on both line segments, as shown in Figure 13(a). However, there might be an intersection of lines in the rendered image, even if $\mathfrak{s}_\mathfrak{A}$ and $\mathfrak{s}_\mathfrak{B}$ do not intersect. To avoid this, the distance ε of the two line segments has to be higher than ε_{min}, where ε_{min} has to be at least one pt larger than 2 times the *Pen Shape* widths w_P. Let $\mathbf{w}$ be the point of the opposite line segment of $\mathbf{l}_\varepsilon \in \{\mathbf{l}'^i_\mathfrak{A}, \mathbf{l}'^{i+1}_\mathfrak{A}, \mathbf{l}'^i_\mathfrak{B}, \mathbf{l}'^{i+1}_\mathfrak{B}\}$ that is closest to $\mathbf{l}_\varepsilon$, and then we get $\varepsilon = \|\mathbf{l}_\varepsilon - \mathbf{w}\|$. In case that $\mathbf{z}$ lies only on one line segment $\mathfrak{s}_{\mathfrak{B}/\mathfrak{A}}$, we know that $\mathbf{l}_\varepsilon \in \{\mathbf{l}'^i_{\mathfrak{A}/\mathfrak{B}}, \mathbf{l}'^{i+1}_{\mathfrak{A}/\mathfrak{B}}\}$, as shown in Figure 13(b). Otherwise $\|\mathbf{l}_\varepsilon - \mathbf{w}\|$ must be calculated for all four points.

Outcomes of the described techniques of text page generation are shown in Figure 14. How to determine a proper baseline function $f(x, y_0)$ will be discussed in the next section.

5.6. Optimization of Baseline Functions. A set of baselines Ξ are n sequences of bounding boxes of all PAW within a page, ordered from the first to the last PAW of a line. To validate and optimize synthetic baselines Ξ_s, which are a result of $f(x, y_0)$, we calculate the correlation $\rho = \text{corr}(\Xi_r, \Xi_s)$, where Ξ_r contains 11295 PAW extracted from natural handwritings.

To get the global correlation ρ_g, we train a Gaussian Mixture Model (GMM) on Ξ_r. Therefore, features x as the average μ and sigma σ of the normalized space between text lines, $\mu \wedge \sigma$ of the angle ϕ between a text line and the horizontal, or $\mu \wedge \sigma$ of the change of ϕ depending on y_0 are used. Subsequently, we use the GMM to calculate the log likelihood $\log \mathscr{L}(\theta \mid \mathbf{x})$, where Ξ_s with $\mathbf{x}$ belongs to the class of natural baselines θ. Now the global correlation can be computed by $\rho_g = n^-1 \sum^n \log \mathscr{L}(\theta \mid \mathbf{x})$.

To detect lines that have odd curvatures, we also compare each synthetic line with all natural ones. Therefore, we represent each line by $\mathfrak{L}$, which is a series of the geometrical centers $\mathfrak{c}$ of the PAW-bounding boxes. To ease the comparison, all natural $\mathfrak{L}_r$ and synthetic $\mathfrak{L}_s$ lines are normalized and translated, so that their first (rightmost) $\mathfrak{c}$ lies within the origin. For all m_Ξ centers $\mathfrak{c} \in \mathfrak{L}_s$ we search neighbored $\mathfrak{c}^a, \mathfrak{c}^b \in \mathfrak{L}_r$ that fulfill $\mathfrak{c}^a_x > \mathfrak{c}_x > \mathfrak{c}^b_x$. Now $\rho_l = \text{corr}(\mathfrak{L}_s, \mathfrak{L}_r)$ can be calculated:

$$\rho_l = \log\left(1 - \left(\frac{\sum^{m_\Xi} \|\mathfrak{c} - \mathfrak{c}^r\|}{m_\Xi}\right)\right),$$
$$\mathfrak{c}^r = \frac{\mathfrak{c}^a_x - \mathfrak{c}_x}{\mathfrak{c}^b_x - \mathfrak{c}^a_x} \mathfrak{c}^a + \frac{\mathfrak{c}_x - \mathfrak{c}^b_x}{\mathfrak{c}^b_x - \mathfrak{c}^a_x} \mathfrak{c}^b. \tag{14}$$

Using the average $\overline{\rho}_l$ of the five best matches ρ_l we get $\rho = \overline{\rho}_l + \rho_g$, where ρ indicated how proper $f(x, y_0)$ simulates the shapes of natural baselines. The functions f_i have to be defined manually within the UI; however, an automatically optimization can be initialized subsequently. We defined multiple f_i, reflecting different peculiarities that could be observed studying historical and other Arabic handwritings. The function that defines a set of ground truth (nonhistorical) text pages of the IESK-arDB and that was used to generate the syntheses in Figure 14 is $f_1(x, y_0)$:

$$-2q_1 x y_0 + y_0 \left[0.7 \sin(2.5x - 1.2) + q_2 \cdot \sin(10x) + q_3 \cdot \sin(8x + 2)\right]. \tag{15}$$

The parameters q_i are redefined by Gaussian distributions $\mathcal{N}_i(\mu_i, \sigma_i)$ for each page synthesis. To find the optimal q_i, we use genetic programming where μ_i, σ_i are the genetic representation and ρ is the fitness of an individual. For f_1 we get the parameterization:

$$q_1 \Longleftarrow \mathcal{N}(0.15, 0.84),$$
$$q_2 \Longleftarrow \mathcal{N}(0.67, 0.51), \tag{16}$$
$$q_3 \Longleftarrow \mathcal{N}(0.35, 0.17).$$

Depending on the defined formula and features of the used Ξ_r, more or less challenging page syntheses can be achieved.

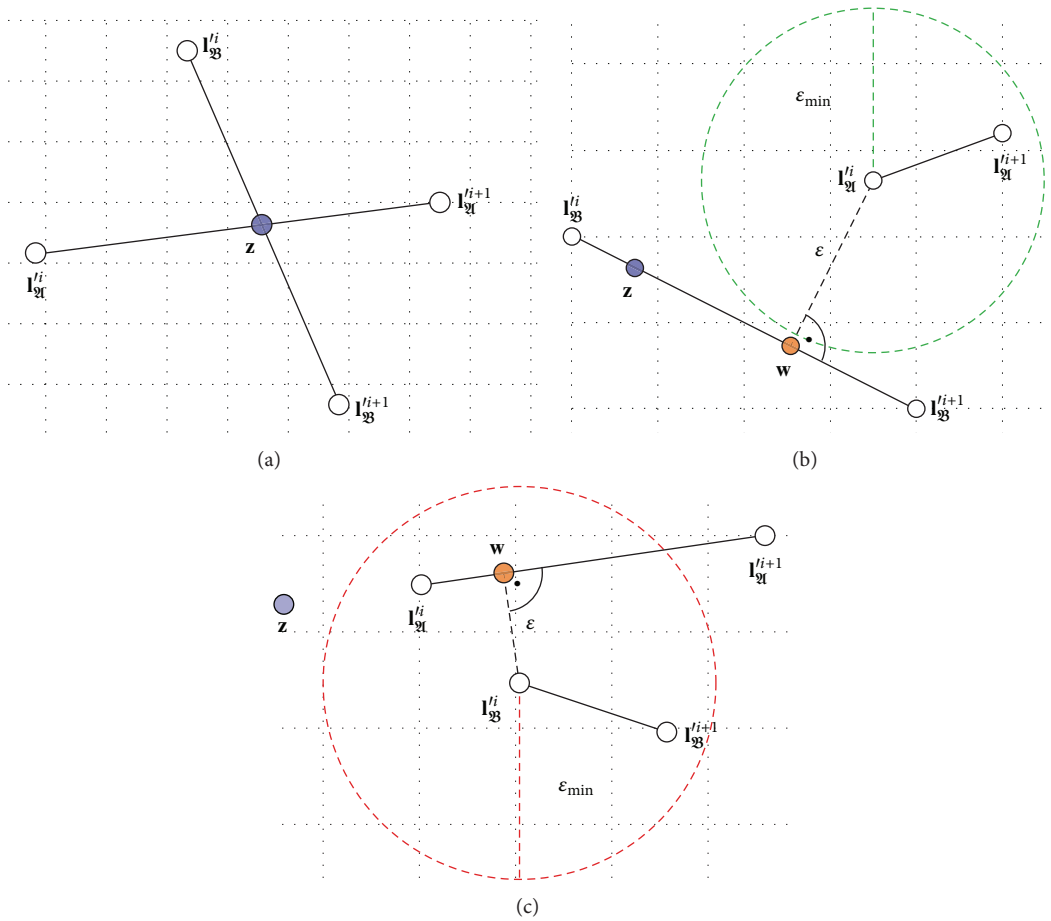


FIGURE 13: Detecting intersections of two line segments, where the grid has the size 1 pt to suggest the synthesized image, ε is 2 pt. (a) The lines segments intersect. (b) The line segments are close and $\mathfrak{s}_{\mathfrak{B}}$ points to $\mathfrak{s}_{\mathfrak{B}}$, but there is no need to move $\mathfrak{A}$ since their shortest distance ε is higher than $\varepsilon_{\min}$. (c) The intersection point $\mathbf{z}$ is outside both line segments, but ε is smaller than $\varepsilon_{\min}$, so an intersection of the corresponding lines of the synthesized image is expected.

6. Results and Discussions

We found that our system is able to synthesize multiple realistic samples for all words that do not include special characters like Hamza over Nabira or ligatures like LamAlif or digits (which can be included when extending the letter database). In the following we propose a method to declare suitable functions for Baseline definitions and validate the applicability of our synthesis outcomes.

6.1. Synthesis Evaluation. Since the data synthesis module is built to ease the development and validation of methods that are related to document analysis, it is not only of interest whether syntheses look realistic or not. In fact it is crucial how image processing methods behave when being fed with synthetic instead of natural data. This is investigated in this section, where a method that segments handwritten Arabic words into letters is validated on such data. We chose word segmentation as example, due to its sensitivity to character shapes as well as global features like overlapping PAWs or varying Kashida length.

Segmentation of Arabic Words. The segmentation method, which is used for the following experiments, is described in [4]. It is based on topological features and a set of rules that reduces all candidates to a final set of points, which divide two neighbored letters. In contrast to other approaches, candidates are not minima that indicate the middle of a Kashida, but typically the following branch point.

Comparison of Real and Synthetic Validation Databases. The synthetic samples which are created by the proposed approach are meant as training or testing data for different document analysis methods. To investigate whether these syntheses can be used instead or additional to natural samples, we created synthetic samples (png files + ground truth) of all words of the IESK-arDB database [4] that we call IESK-arDB-Syn in the following. We validate the described segmentation method on both databases using

(a) Real (b) Synthesis (PDF) (c) Synthesis (PNG)

(d) (e) (f)

FIGURE 14: Examples for syntheses of complete text pages: (a) one of the natural handwritings that have been used to compute function f; (b) synthesis using optimized f_1 (given by (15)), rendered as PDF; (c) another synthesis using f_1, rendered as png image. More examples inclusive ground truth will be added at http://www.iesk-ardb.ovgu.de/ soon. Text synthesis from a BBC newsletter (d); BBC article about a virus outbreak in Saudi Arabia [5] (e-f); two syntheses of the given paragraph of (d).

TABLE 4: Validation of the segmentation method.

Database	IESK-arDB		IESK-arDB-Syn	
Number of word samples	2540		9000	
Measure	μ	σ	μ	σ
Error per word	1.67	0.13	1.74	0.024
Error per letter	0.35	0.026	0.34	0.0045
Over segmentation (per word)	0.79	0.097	0.86	0.0066
Under segmentation (per word)	0.87	0.071	0.88	0.019
Perfect segmentation (per word)	0.17	0.0019	0.13	0.0067

cross validation. As shown in Table 4 the detected error rates are comparable. This proves at least that the proposed synthesis method is capable of reflecting those features of natural Arabic words, which are critical for segmentations.

Furthermore the proposed synthesis approach enables investigating the robustness of the segmentation method against the influence of particular features. Therefore, we built modifications of IESK-arDB-Syn for experiments A and B, using the UI to ensure that only the investigated feature differs for samples of the same writer. The results of these experiments are shown in Figure 15.

Experiment A: *ASM Eigenvector Intensity*. The intensities c_i of the used eigenvectors define the similarity of a computed letter shape $\mathbf{u}$ and $\bar{\mathbf{x}}$, where maximal intensities (c_i) often causes unexpected, deformed shapes which are hard to classify. The experiment confirms that eigenvector intensities are also proportional to segmentation error frequency. However, even strongly deformed letter shapes reduce the segmentation results only moderately, since their influence on Kashidas and key features, as branch points, is less than their influence on the signature of the letter curvature, which seems more vital regarding character recognition.

Experiment B: *Writer*. As one can see in Figure 15(b), the segmentation method is quite sensitive to the writer dependent style. Best performance is achieved on writer 1, since letters are written in a proper style and have long Kashidas.

Experiment C: *Skew*. Although skew correction is a common preprocessing step, this experiments show that the used segmentation method is robust against a skew of $\pm 20^\circ$.

Experiment D: *Kashida Length*. Kashidas are the connections of two letters, which have a high variation in case of Arabic handwritings. The experiment shows that a valid segmentation is especially difficult in case of very small Kashidas, since most structures that indicate a potential dividing point are hidden or vanished in such cases. In extreme cases neighbored letters can be written one above the other. This effect could be observed in both natural and synthetic handwritings and makes segmentation very challenging. In contrast to slant and skew variation, Kashida related problems cannot be solved by a simple preprocessing technique.

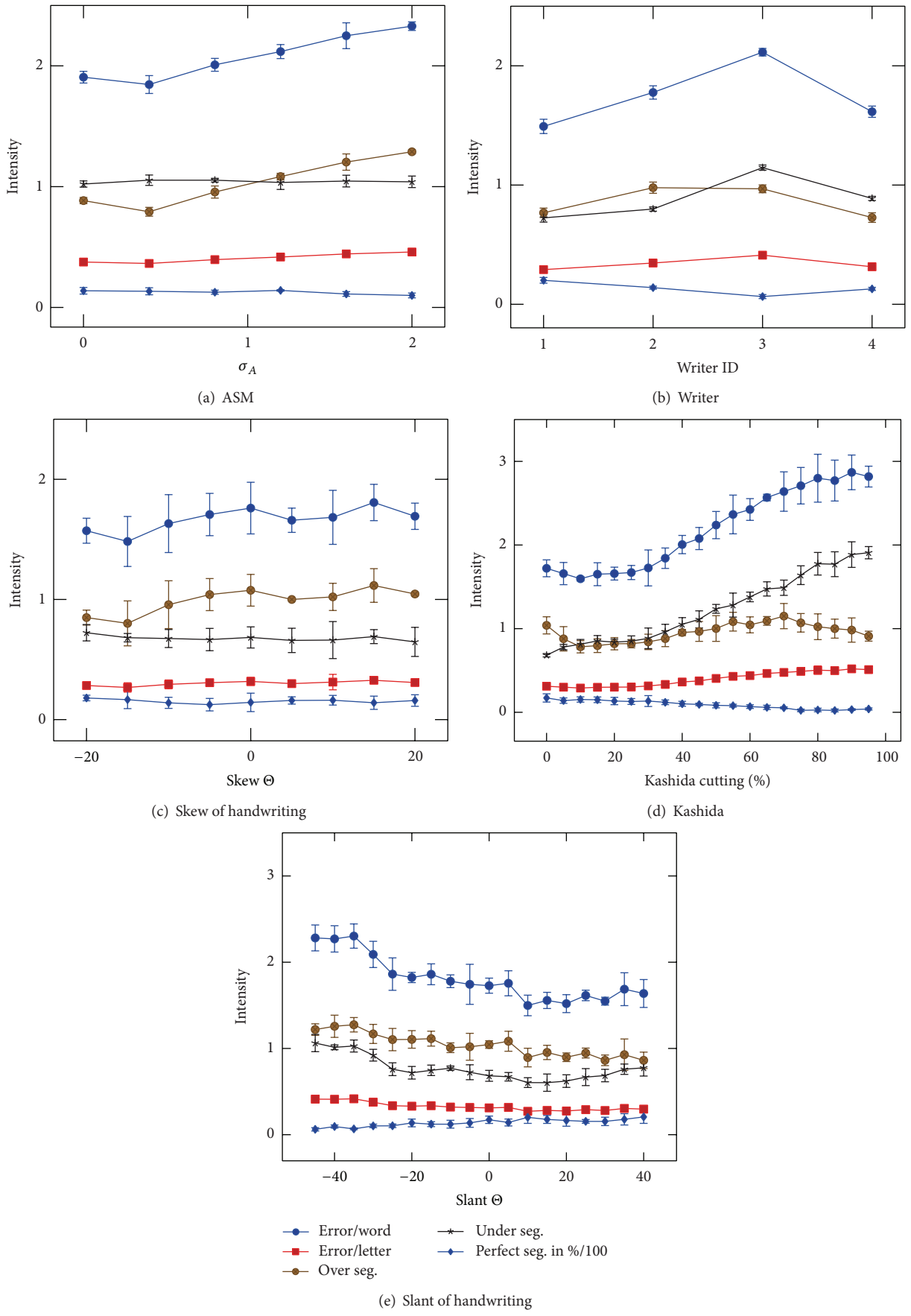


Figure 15: Experiments show the influence of different features on the average error of the word segmentation method.

Experiment E: *Slant*. The used segmentation method does only segment at rows with exactly one foreground pixel. Hence, extreme slants can cause segmentation errors, if the slant causes strongly overlapping ascenders. The experiment shows that a slant of about 20° even slightly improves the segmentation results, which might be caused by the frequent appearance of Alif (ا) in end-form that can be detected more reliable, if it has a positive slant. This effect is weakened when reducing the Kashidas length, though. Finally, the experiment shows that slant correction is not a mandatory but nonetheless useful preprocessing step for the proposed segmentation method, especially for handwritings with strong negative slant.

7. Conclusion

We have presented an efficient approach to generate pseudo handwritten Arabic words and text pages, including diacritic marks (dots), from Unicode. Online sample and Active Shape Model based glyphs from multiple writers as well as affine transformations allow to generate various images for a given Unicode string to cover the variability of human handwritings. Data of new writers can be added easily and efficiently, since the definition of manual landmarks is not necessary. Features as the slant can be controlled manually if desired. Interpolation methods and a rendering technique are used to meet the properties of offline handwritings. We investigated the practical applicability of the synthesis by validating a segmentation algorithm on natural and synthetic data getting comparable results.

In our future work we are going to extend the used alphabet, acquire letter samples from more writers, and reduce the amount of synthesized words by clustering techniques as affinity propagation. This will help to synthesize compact but representative databases and use them to train and test methods for handwriting recognition and document analysis approaches.

Conflict of Interests

The authors declare that they have no competing interests.

Acknowledgment

The authors would like to extend their sincere appreciation to the Deanship of Scientific Research at King Saud University for its funding of this International Research Group (IRG14-28).

References

[1] V. Märgner and H. El Abed, "Databases and competitions: strategies to improve Arabic recognition systems," in *Arabic and Chinese Handwriting Recognition: SACH 2006 Summit College Park, MD, USA, September 27-28, 2006 Selected Papers*, vol. 4768 of *Lecture Notes in Computer Science*, pp. 82–103, Springer, Berlin, Germany, 2008.

[2] Y. Elarian, R. Abdel-Aal, I. Ahmad, M. T. Parvez, and A. Zidouri, "Handwriting synthesis: classifications and techniques," *International Journal on Document Analysis and Recognition*, vol. 17, no. 4, pp. 455–469, 2014.

[3] Y. Elarian, H. A. Al-Muhsateb, and L. M. Ghouti, "Arabic handwriting synthesis," in *Proceedings of the 1st International Workshop on Frontiers in Arabic Handwriting Recognition*, 2011, http://hdl.handle.net/2003/27562.

[4] M. Elzobi, A. Al-Hamadi, Z. Al Aghbari, and L. Dings, "IESK-ArDB: a database for handwritten Arabic and an optimized topological segmentation approach," *International Journal on Document Analysis and Recognition*, vol. 16, no. 3, pp. 295–308, 2013.

[5] http://www.bbc.co.uk/arabic/scienceandtech/.

[6] L. M. Lorigo and V. Govindaraju, "Offline arabic handwriting recognition: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 5, pp. 712–724, 2006.

[7] J. M. Hollerbach, "An oscillation theory of handwriting," *Biological Cybernetics*, vol. 39, no. 2, pp. 139–156, 1981.

[8] G. Gangadhar, D. Joseph, and V. S. Chakravarthy, "An oscillatory neuromotor model of handwriting generation," *International Journal on Document Analysis and Recognition*, vol. 10, no. 2, pp. 69–84, 2007.

[9] R. Plamondon, W. Guerfali, and Scribens (Laboratory), *The Generation of Handwriting with Delta-Lognormal Synergies*, Rapport Technique, Laboratoire Scribens, Département de Génie Électrique, École Polytechnique de Montréal, 1996, http://books.google.de/books?id=6Vn9MwEACAAJ.

[10] I. Guyon, "Handwriting synthesis from handwritten glyphs," in *Proceedings of the 5th International Workshop on Frontiers of Handwriting Recognition*, pp. 309–312, 1996.

[11] T. Varga and H. Bunke, "Perturbation models for generating synthetic training data in handwriting recognition," in *Machine Learning in Document Analysis and Recognition*, S. Marinai and H. Fujisawa, Eds., vol. 90, pp. 333–360, Springer, Berlin, Germany, 2008.

[12] Y. Zheng and D. S. Doermann, "Handwriting matching and its application to handwriting synthesis," in *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR '05)*, pp. 861–865, IEEE Computer Society, Seoul, Republic of Korea, September 2005.

[13] P. Viswanath, N. Murty, and S. Bhatnagar, "Overlap pattern synthesis with an efficient nearest neighbor classifier," *Pattern Recognition*, vol. 38, no. 8, pp. 1187–1195, 2005.

[14] H. I. Choi, S.-J. Cho, and J. H. Kim, "Generation of handwritten characters with bayesian network based on-line handwriting recognizers," in *Proceedings of the 7th International Conference on Document Analysis and Recognition*, pp. 995–999, IEEE, Edinburgh, UK, August 2003.

[15] J. Dolinsky and H. Takagi, "Synthesizing handwritten characters using naturalness learning," in *Proceedings of the IEEE International Conference on Computational Cybernetics (ICCC '07)*, pp. 101–106, Gammarth, Tunisia, October 2007.

[16] S. Al-Zubi, "Active shape structural model," Tech. Rep., 2004.

[17] D. Shi, S. R. Gunn, and R. I. Damper, "Handwritten Chinese radical recognition using nonlinear active shape models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 2, pp. 277–280, 2003.

[18] M. Helmers and H. Bunke, "Generation and use of synthetic training data in cursive handwriting recognition," in *Pattern Recognition and Image Analysis: Proceedings of the 1st Iberian*

Conference, IbPRIA 2003, Puerto de Andratx, Mallorca, Spain, JUne 4–6, 2003, F. J. Perales, A. J. C. Campilho, N. P. de la Blanca, and A. Sanfeliu, Eds., vol. 2652 of *Lecture Notes in Computer Science*, pp. 336–345, Springer, Berlin, Germany, 2003.

[19] C. V. Jawahar, A. Balasubramanian, M. Meshesha, and A. M. Namboodiri, "Retrieval of online handwriting by synthesis and matching," *Pattern Recognition*, vol. 42, no. 7, pp. 1445–1457, 2009.

[20] P. Rao, "Shape vectors: an efficient parametric representation for the synthesis and recognition of hand script characters," *Sadhana*, vol. 18, no. 1, pp. 1–15, 1993.

[21] Y. Xu, H. Shum, J. Wang, and C. Wu, "Learning-based system and process for synthesizing cursive handwriting," US Patent 7,227,993, 2007, http://www.google.co.in/patents/US7227993.

[22] Z. Lin and L. Wan, "Style-preserving English handwriting synthesis," *Pattern Recognition*, vol. 40, no. 7, pp. 2097–2109, 2007.

[23] J. Wang, C. Wu, Y.-Q. Xu, and H.-Y. Shum, "Combining shape and physical models for online cursive handwriting synthesis," *International Journal on Document Analysis and Recognition*, vol. 7, no. 4, pp. 219–227, 2005.

[24] A. O. Thomas, A. Rusu, and V. Govindaraju, "Synthetic handwritten CAPTCHAs," *Pattern Recognition*, vol. 42, no. 12, pp. 3365–3373, 2009.

[25] H. Miyao and M. Maruyama, "Virtual example synthesis based on PCA for off-line handwritten character recognition," in *Document Analysis Systems VII*, vol. 3872, pp. 96–105, Springer, Berlin, Germany, 2006.

[26] T. Varga and H. Bunke, "Generation of synthetic training data for an HMM-based handwriting recognition system," in *Proceedings of the 7th International Conference on Document Analysis and Recognition*, vol. 1, pp. 618–622, Edinburgh, UK, August 2003.

[27] T. Varga, D. Kilchhofer, and H. Bunke, "Template-based synthetic handwriting generation for the training of recognition systems," in *Proceedings of the 12th Conference of the International Graphonomics Society*, pp. 206–211, 2005.

[28] B. B. Chaudhuri and A. Kundu, "Synthesis of individual handwriting in bangla script," in *Proceedings of the ICFHR*, 2008, http://www.cenparmi.concordia.ca/ICFHR2008/Proceedings/papers/cr1079.pdf.

[29] V. Margner and M. Pechwitz, "Synthetic data for Arabic OCR system development," in *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pp. 1159–1163, Seattle, Wash, USA.

[30] Y. Elarian, I. Ahmad, S. Awaida, W. G. Al-Khatib, and A. Zidouri, "An Arabic handwriting synthesis system," *Pattern Recognition*, vol. 48, no. 3, pp. 849–861, 2015.

[31] R. M. Saabni and J. A. El-Sana, "Comprehensive synthetic Arabic database for on/off-line script recognition research," *International Journal on Document Analysis and Recognition*, vol. 16, no. 3, pp. 285–294, 2013.

[32] L. Dinges, M. Elzobi, A. Al-Hamadi, and Z. Al-Aghbari, "Synthizing handwritten arabic text using active shape models," in *Image Processing and Communications Challenges 3*, vol. 102 of *Advances in Intelligent and Soft Computing*, pp. 401–408, Springer, Berlin, Germany, 2011.

[33] L. Dinges, A. Al-Hamadi, and M. Elzobi, "An approach for arabic handwriting synthesis based on active shape models," in *Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR '13)*, pp. 1260–1264, August 2013.

[34] H. Almuallim and S. Yamaguchi, "A method of recognition of arabic cursive handwriting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 715–722, 1987.

[35] P. Xiu, L. Peng, X. Ding, and H. Wang, "Offline handwritten arabic character segmentation with probabilistic model," in *Document Analysis Systems VII*, vol. 3872 of *Lecture Notes in Computer Science*, No. project 60472002, pp. 402–412, Springer, Berlin, Germany, 2006.