COPPE
UFRJ
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia

# JCHAR: JAPANESE CHARACTER HANDWRITING ANALYZER

Fernando de Mesentier Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Ricardo Guerra Marroquim

Rio de Janeiro
Julho de 2014

JCHAR: JAPANESE CHARACTER HANDWRITING ANALYZER

Fernando de Mesentier Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Ricardo Guerra Marroquim, D.Sc.

Prof. Claudio Esperança, Ph.D.

Prof. Luiz Henrique de Figueiredo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2014

Silva, Fernando de Mesentier

jCHAR: Japanese Character Handwriting Analyzer/Fernando de Mesentier Silva. – Rio de Janeiro: UFRJ/COPPE, 2014.

X, 40 p.: il.; 29, 7cm.

Orientador: Ricardo Guerra Marroquim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 39 – 39.

1. Sketching. 2. Stroke analysis. 3. Japanese language. I. Marroquim, Ricardo Guerra. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

JCHAR: JAPANESE CHARACTER HANDWRITING ANALYZER

Fernando de Mesentier Silva

Julho/2014

Orientador: Ricardo Guerra Marroquim

Programa: Engenharia de Sistemas e Computação

Apresenta-se, nesta dissertação, uma ferramenta de sketching voltada para auxiliar aqueles que buscam o aprendizado da língua escrita japonesa através de uma interface que disponibiliza feedback pontual sobre caracteres escritos à mão livre. O usuário escolhe o caractere que ele tentará emular e então o escreve utilizando traços à mão livre. O sistema analisa os traços e compara a entrada a um template exato, e então faz sugestões de mudanças que devem ser feitas em aspectos chave para melhorar a escrita do usuário. A análise do caractere envolve uma bateria de testes com os traços, além da construção de uma hierarquia de erros em uma tentativa de identificar erros que levam a outros. Mesmo que o método tenha sido aplicado diretamente à caligrafia japonesa, ele é extensível a outros tipos de caracteres e aplicações.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

# JCHAR: JAPANESE CHARACTER HANDWRITING ANALYZER

Fernando de Mesentier Silva

July/2014

Advisor: Ricardo Guerra Marroquim

Department: Systems Engineering and Computer Science

In this work we present a sketching tool that aims at helping those seeking to learn the Japanese written language through an easy-to-use interface that provides punctual feedback over handwritten characters. The user chooses the character that he will try to emulate and then writes it using free hand strokes. The system proceeds by analyzing the strokes and comparing the input to a ground truth template, and then gives suggestions on key aspects that need correction to improve the user's writing. The character analysis involves a battery of tests with the strokes and building an error hierarchy in an attempt to trace what error leads to another. Even though the method is directly applied to Japanese calligraphy, it is extendable to other types of characters and applications.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The process of learning to write Japanese characters by hand involves, traditionally, a lot of repetitions that usually need to be made under the supervision of a teacher, as characters possesses features that are restrained by a few rules.

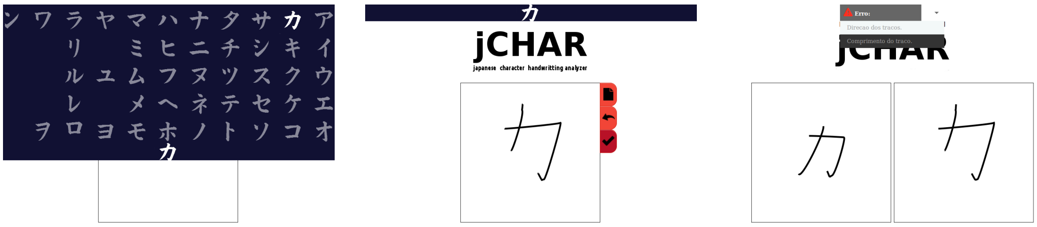
The Japanese language is composed by two syllabaries and an alphabet. Among these, characters share many characteristics in their structure. Features such as stroke order and orientation need to be stressed over the process of learning as they play an important role in the written language.

Sketching interfaces have as purpose introducing a more natural way of interacting with a machine. Sketch-based interfaces are usually simpler than traditional ones, as the freehand sketching input overcomes the need of numerous menus and buttons, but although more simplistic in its interaction it makes for a much harder implementation. Input interfaces usually introduce noise and have differences in sampling ratios, as they capture position over fixed time-frames, faster drawings have fewer sampling points than slower ones. So, it is necessary to refine input to produce quality data.

There are a lot of subtleties in analyzing handwriting. Some differences in representation are usually allowed in character writing, but incorporating this concept in a computational model is a complex task. Quality feedback is of most importance, as it can make the process of learning easier and, consequently, more pleasuring.

In this work we present a method to evaluate the user's hand-drawn input character in order to provide substantial and punctual feedback. For our tests we used the characters in the Katakana alphabet. The application evaluates the input by comparing its most important features against a template raising a set of possible corrections to be made. To avoid confusing, or even intimidating, the user, since a too straightforward approach might find a high number of errors, we propose a method to create a hierarchy to prioritize and select the most relevant errors in order to provide the most helpful feedback possible.

Our application starts with the user selecting a character and drawing it. The


ン ワ ラ ヤ マ ハ ナ タ サ カ ア
リ ミ ヒ ニ チ シ キ イ
ル ユ ム フ ヌ ツ ス ク ウ
レ メ ヘ ネ テ セ ケ エ
ヲ ロ ヨ モ ホ ノ ト ソ コ オ
カ
カ
jCHAR
japanese character handwritting analyzer
Erro:
Direcao dos tracos.
Comprimento do traco.
jCHAR

(a) Character menu (b) Canvas with a drawing (c) Error feedback menu

Figure 1.1: Overview

input drawing is transformed to the same space as the template. Comparison is then done following the important features present in the Japanese character construction. Special types of errors detected in the drawing are then sorted by the priority calculated through a hierarchy algorithm. At last, visual feedback is provided to the user [Figure 1.1].

The manual selection could also be replaced by an automatic character recognition before the analyses, however, this is out of the scope of this dissertation. Even though we only present one syllabary here, our work can be extended across the entire Japanese written language, or any other set of characters with similar restrictions and properties, given the necessary templates.

We present related work in chapter 2. In chapter 3 we give an introduction to the Japanese written language. In chapter 4 we comment on our previous work. In chapter 5 we discuss about the steps taken to process input. In chapter 6 we present the method used for the feature analyses. In chapter 7 we detail our implementation. In chapter 8 we show and discuss our experiments. In chapter 9 we give our conclusions. In chapter 10 we present some future works. In appendix A we list the parameters and values used in the implementation.

# Chapter 2

# Related work

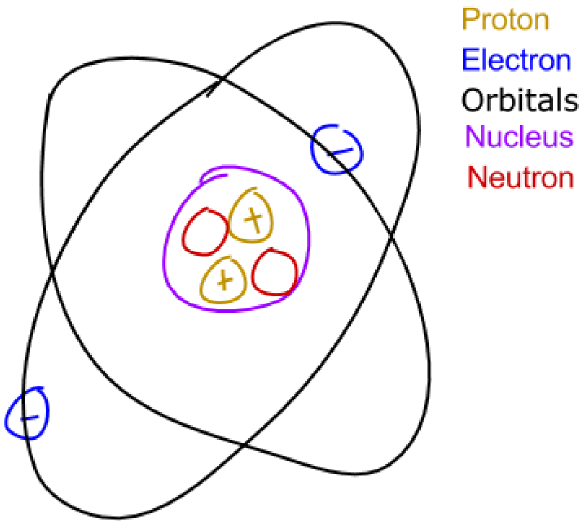


Figure 2.1: An example of a representation in sKEA. In this drawing the color of the labels matches the color of the glyphs that they are associated with.

In [1] sKEA(sketching Knowledge Entry Associate) is presented. The purpose of sKEA is to be a sketching understanding system. The user can make drawings and give them meanings to build the sketch intended. By drawing a collection of ink strokes, called glyphs, a visual representation is created, then the user gives it meaning by classifying it as an Entity or Relationship and labeling with a concept present in the sKEA knowledge database [Figure 2.1]. A sketch can also be compared, and this comparison analyzed, inside the sKEA interface. Comparison is done using the Structure-Mapping Engine and is based both on the visual and conceptual material of the sketches. The sKEA approach is applied in the nuSketch [2] architecture. Its architecture uses the concepts of working on the spatial relationships that was present in sKEA, with the visual structure of the sketch being denoted by 5 attributes: groupings, positional, relationship, size and orientation. These attributes, together with the conceptual representation extracted from the labeling, drive the sense of similarity present in this approach. Although a character learning scheme could be incorporated in this system, we felt that a more automated and interactive approach could prove to be beneficial.

An approach closer to ours is the one in CogSketch [3], a system based on the architecture presented in nuSketch. As the previous work it is based upon, CogSketch works to achieve a more general solution for sketch analysis. With the implementation of the sKEA metalayer interface that can express the relationship between multiple sketches, and the automatic edge segmentation of a glyph to enable computation of the qualitative relationship of these edges, CogSketch shows its potential as a platform for educational software. An experiment of a dedicated educational interface for CogSketch is presented, where one user can write a problem description and present its solution. Other users will then try to emulate that solution through the description of the problem, using the set of labels present, with the platform giving feedback based on the analogies of both the submitted and actual solution that are extracted from the differences in the important relationships. For example, the user could be asked to draw an atom diagram, with the solution being the sketch on Figure 2.1, the user would be limited to the labels present in the solution, i.e., Proton, Electron, Orbitals, Nucleus and Neutron, and could have feedback on an incorrect relationship, such as drawing the Proton outside the Nucleus. Inputting a problem description/solution pair is still a tedious task as it is necessary to do all the labeling and specification of relationships. Since we work with a finite set of features to generate feedback, our work achieves a more specialized and easy to use solution by removing the need for labeling and providing more punctual feedback.

Mechanix [4] specializes the CogSketch educational approach to free-body and truss diagrams. Following the educational interface that was presented in CogSketch, one user draws a template that represents the solution of a free-body diagram or truss problem and then other users give their input to be compared, evaluated and given feedback about. The system focus on identifying closed shapes that could represent the body or truss that are the main part of the solution, and then comparing those to the ones in the template. Comparison takes into account geometric properties and graphical structure, disallowing transformations such as rotation and reflection as these deviate from the solution previously stated. Our work follows a similar fashion for Japanese characters, with the templates already given and the necessary approach for the particularities that we present in this work.

In "How Do Humans Sketch Objects?"[5] an analysis is made of the human performance on recognizing and categorizing sketches compared to computer recognition techniques. The work defines 250 object categories in which human sketches were acquired to form a database composed of 20,000 sketches. Then, humans are exposed to a number of random sketches from the database, each from a different category, and asked to classify them inside the 250 existing categories. They achieved an average correct recognition rate of 73%, with results varying for the different categories. A feature method is then presented with which a computer, trained with their database, classifies sketches to later identify unknown ones. This method reaches a 56% accuracy. An interesting point the authors make, that exemplifies the difficulties of analyzing hand drawn sketches, is that humans usually express their ideas in free hand sketches without image precision, such as drawing a stick figure to represent a person. However, they are still able to have the meaning of their drawing fully understood by others.

Furthermore, stroke recognition and analysis methods and algorithms were already explored by [6–8]. However, the stroke-by-stroke comparisons presented did not attend our requirements, as it was not enough to provide the feedback we needed to achieve and left some character ambiguities untreated, so we decided to approach a solution that rendered character analysis to a multi-stroke comparison. The 1$ Unistroke Recognizer [6] and the Angle Quantization Fast Stroke Matching [7] are further described in the Character Recognition Chapter of this dissertation.

In the Online Handwritten Kanji Recognition Based on Inter-stroke Grammar [9] a Kanji character recognizer method using a stochastic context-free grammar and recognition of substrokes through Hidden Markov Models is documented. Since Kanjis are composed of multiple parts, the method builds a recursive hierarchical structure of the different parts that compose the characters until it reaches the actual strokes [Figure 2.2]. With the process of character composition being represented in terms of rules, terminal and non-terminal symbols, the stochastic context-free grammar is built. To better distinguish similar characters, relative position between parts that compose the characters is also taken into account when recognizing the handwriting. The work shows that the spatial relationship of the different parts plays an important part in the representation of a character, a concept which guides some of the discussions in this dissertation.

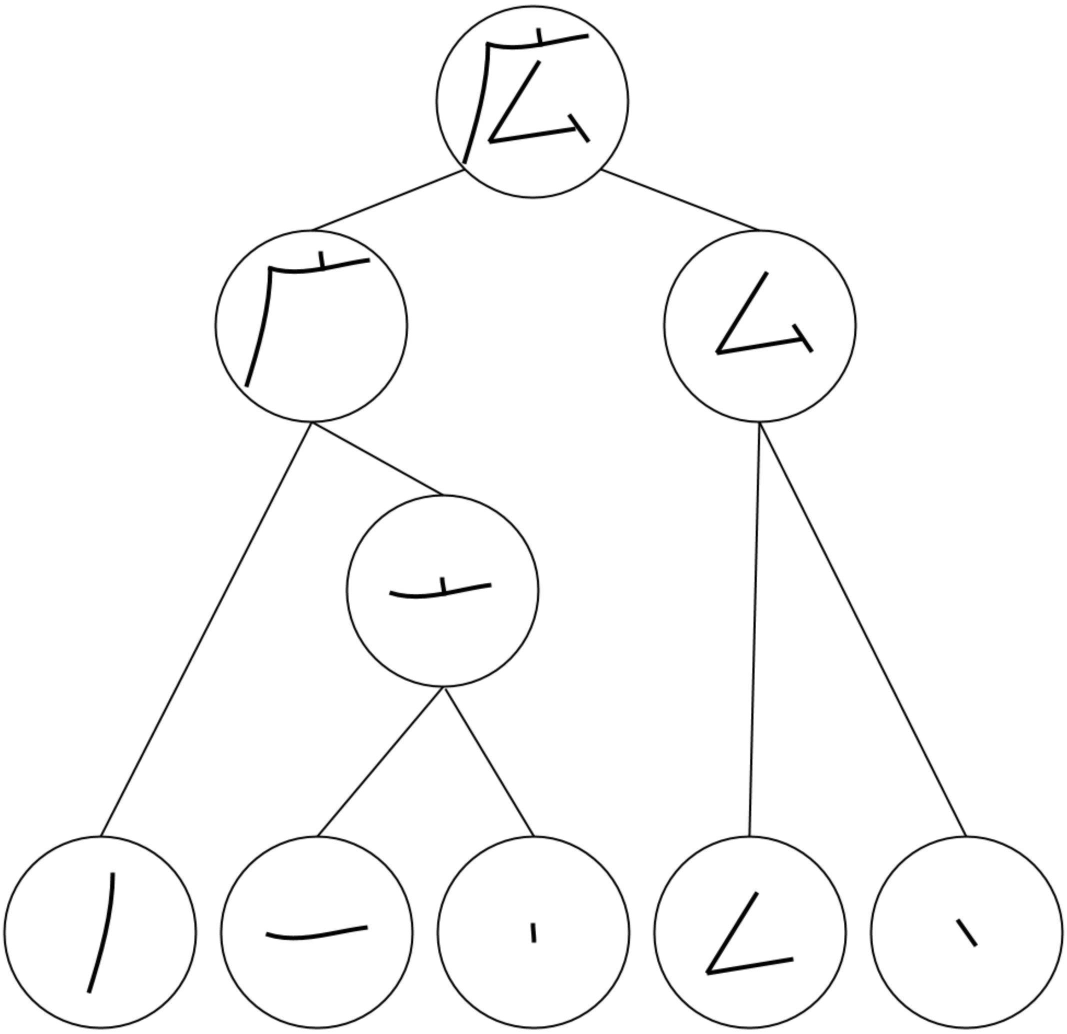
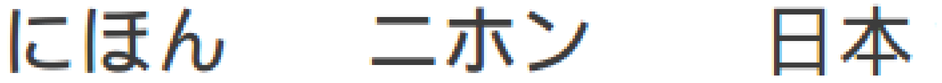


Figure 2.2: Kanji decomposition into its character-parts.

# Chapter 3

# Japanese Written Language



(a) Hiragana (b) Katakana (c) Kanji

Figure 3.1: The word Japan written in the two syllabaries and the Kanji alphabet.

The Japanese written language is composed of 2 syllabaries, Hiragana and Katakana, and the Kanji alphabet [Figure 3.1]. Although it is possible to write every Japanese word without using any Kanji characters, all 3 of them are used together when writing.

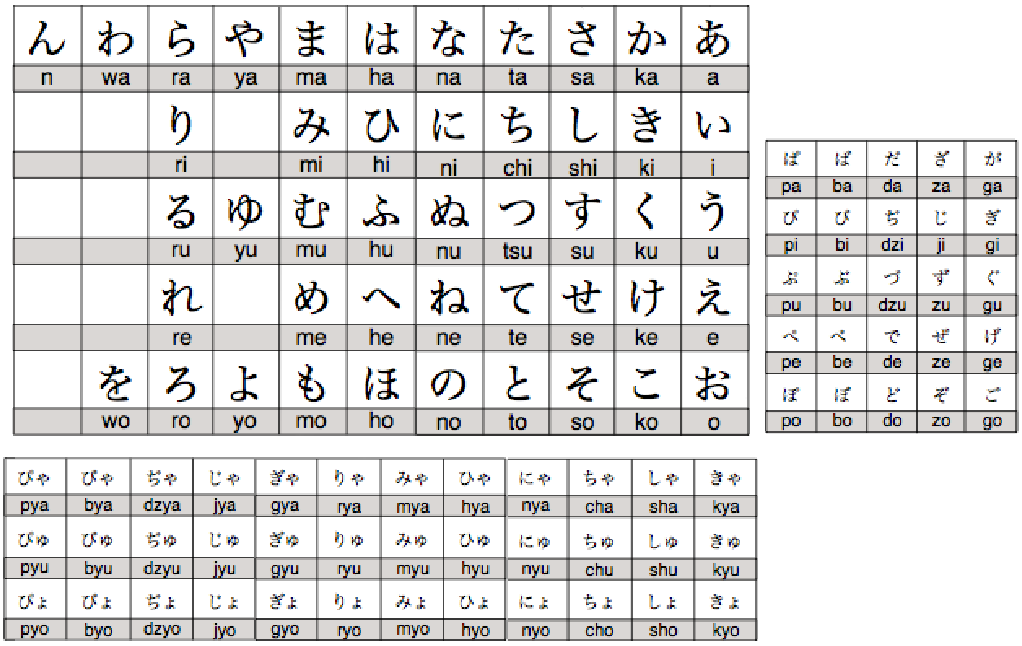
Among all 3 writing groups, the characters follow very similar rules and topology, usually varying only in complexity.

## 3.1 Hiragana

The Hiragana [Figure 3.2] is made up of 46 unique characters. Each character represents one sound in the Japanese language. It's used mainly to write verbs and words to which there is no Kanji.

## 3.2 Katakana

The Katakana [Figure 3.3] has different characters for the same sounds represented in the Hiragana, their difference being the usage. Katakana is mainly used to write foreign words.



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ん | わ | ら | や | ま | は | な | た | さ | か | あ |
| n | wa | ra | ya | ma | ha | na | ta | sa | ka | a |
| | | り | | み | ひ | に | ち | し | き | い |
| | | ri | | mi | hi | ni | chi | shi | ki | i |
| | | る | ゆ | む | ふ | ぬ | つ | す | く | う |
| | | ru | yu | mu | hu | nu | tsu | su | ku | u |
| | | れ | | め | へ | ね | て | せ | け | え |
| | | re | | me | he | ne | te | se | ke | e |
| | を | ろ | よ | も | ほ | の | と | そ | こ | お |
| | wo | ro | yo | mo | ho | no | to | so | ko | o |

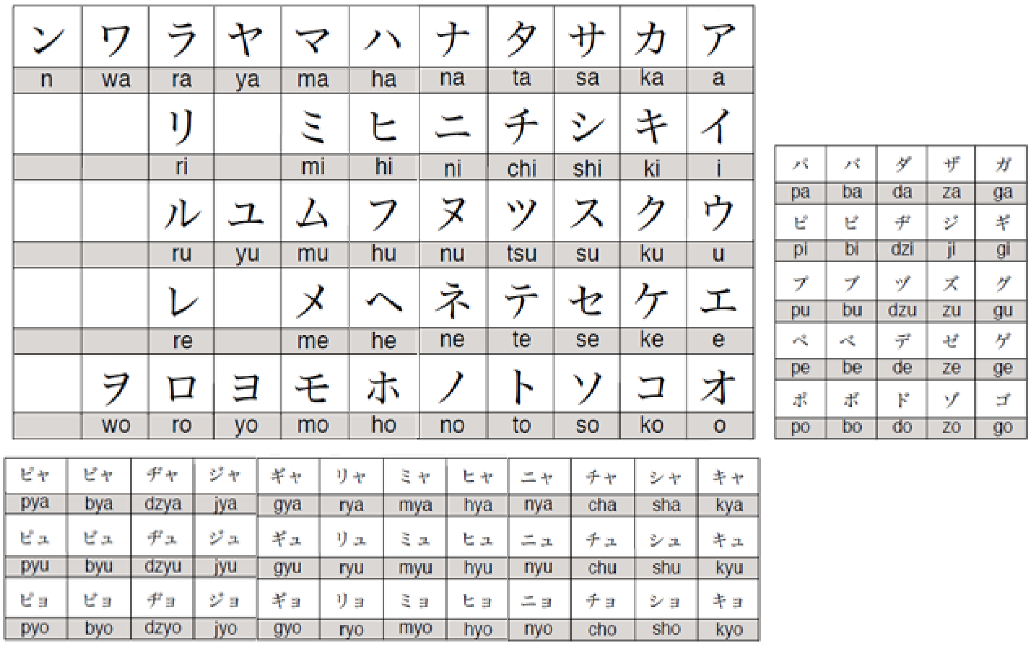
| | | | | |
|---|---|---|---|---|
| ぱ | ば | だ | ざ | が |
| pa | ba | da | za | ga |
| ぴ | び | ぢ | じ | ぎ |
| pi | bi | dzi | ji | gi |
| ぷ | ぶ | づ | ず | ぐ |
| pu | bu | dzu | zu | gu |
| ぺ | べ | で | ぜ | げ |
| pe | be | de | ze | ge |
| ぽ | ぼ | ど | ぞ | ご |
| po | bo | do | zo | go |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ぴゃ | びゃ | ぢゃ | じゃ | ぎゃ | りゃ | みゃ | ひゃ | にゃ | ちゃ | しゃ | きゃ |
| pya | bya | dzya | jya | gya | rya | mya | hya | nya | cha | sha | kya |
| ぴゅ | びゅ | ぢゅ | じゅ | ぎゅ | りゅ | みゅ | ひゅ | にゅ | ちゅ | しゅ | きゅ |
| pyu | byu | dzyu | jyu | gyu | ryu | myu | hyu | nyu | chu | shu | kyu |
| ぴょ | びょ | ぢょ | じょ | ぎょ | りょ | みょ | ひょ | にょ | ちょ | しょ | きょ |
| pyo | byo | dzyo | jyo | gyo | ryo | myo | hyo | nyo | cho | sho | kyo |

Figure 3.2: Hiragana. *Original image taken from http://swordartonline.wikia.com/ on June/16/2014 and is licensed under public domain.*

## 3.3 Kanji

The Kanji [Figure 3.4] is made up of Chinese characters that were adopted by Japan. It has over 50000 different characters. A single Kanji can represent one or more different words, and may have more than one pronunciation. Two or more Kanjis can also be put together to compose a single word. The majority of the Kanji characters are more complex, having up to 60+ strokes, when compared with the Hiragana and Katakana characters, which are drawn with 4 strokes maximum.

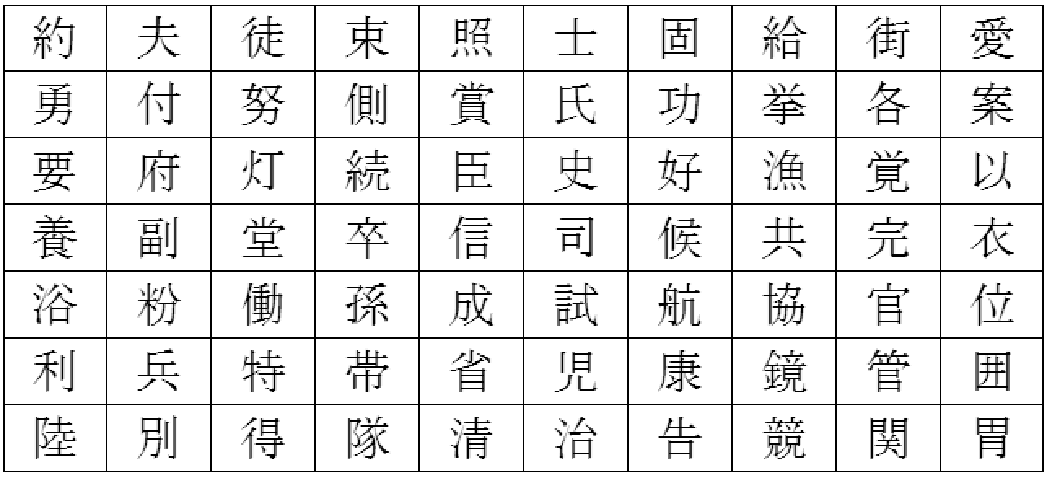


| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ン | ワ | ラ | ヤ | マ | ハ | ナ | タ | サ | カ | ア |
| n | wa | ra | ya | ma | ha | na | ta | sa | ka | a |
| | | リ | | ミ | ヒ | ニ | チ | シ | キ | イ |
| | | ri | | mi | hi | ni | chi | shi | ki | i |
| | | ル | ユ | ム | フ | ヌ | ツ | ス | ク | ウ |
| | | ru | yu | mu | hu | nu | tsu | su | ku | u |
| | | レ | | メ | ヘ | ネ | テ | セ | ケ | エ |
| | | re | | me | he | ne | te | se | ke | e |
| | ヲ | ロ | ヨ | モ | ホ | ノ | ト | ソ | コ | オ |
| | wo | ro | yo | mo | ho | no | to | so | ko | o |

| | | | | |
|---|---|---|---|---|
| パ | バ | ダ | ザ | ガ |
| pa | ba | da | za | ga |
| ピ | ビ | ヂ | ジ | ギ |
| pi | bi | dzi | ji | gi |
| プ | ブ | ヅ | ズ | グ |
| pu | bu | dzu | zu | gu |
| ペ | ベ | デ | ゼ | ゲ |
| pe | be | de | ze | ge |
| ポ | ボ | ド | ゾ | ゴ |
| po | bo | do | zo | go |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ピャ | ビャ | ヂャ | ジャ | ギャ | リャ | ミャ | ヒャ | ニャ | チャ | シャ | キャ |
| pya | bya | dzya | jya | gya | rya | mya | hya | nya | cha | sha | kya |
| ピュ | ビュ | ヂュ | ジュ | ギュ | リュ | ミュ | ヒュ | ニュ | チュ | シュ | キュ |
| pyu | byu | dzyu | jyu | gyu | ryu | myu | hyu | nyu | chu | shu | kyu |
| ピョ | ビョ | ヂョ | ジョ | ギョ | リョ | ミョ | ヒョ | ニョ | チョ | ショ | キョ |
| pyo | byo | dzyo | jyo | gyo | ryo | myo | hyo | nyo | cho | sho | kyo |

Figure 3.3: Katakana. *Original image taken from http://swordartonline.wikia.com/ on June/16/2014 and is licensed under public domain.*



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 約 | 夫 | 徒 | 束 | 照 | 士 | 固 | 給 | 街 | 愛 |
| 勇 | 付 | 努 | 側 | 賞 | 氏 | 功 | 挙 | 各 | 案 |
| 要 | 府 | 灯 | 続 | 臣 | 史 | 好 | 漁 | 覚 | 以 |
| 養 | 副 | 堂 | 卒 | 信 | 司 | 候 | 共 | 完 | 衣 |
| 浴 | 粉 | 働 | 孫 | 成 | 試 | 航 | 協 | 官 | 位 |
| 利 | 兵 | 特 | 帯 | 省 | 児 | 康 | 鏡 | 管 | 囲 |
| 陸 | 別 | 得 | 隊 | 清 | 治 | 告 | 競 | 関 | 胃 |

Figure 3.4: Example of a few Kanjis. *Original image taken from http://www.learn-japanese.info/ on June/16/2014 and is licensed under public domain.*

# Chapter 4

# Character Recognition

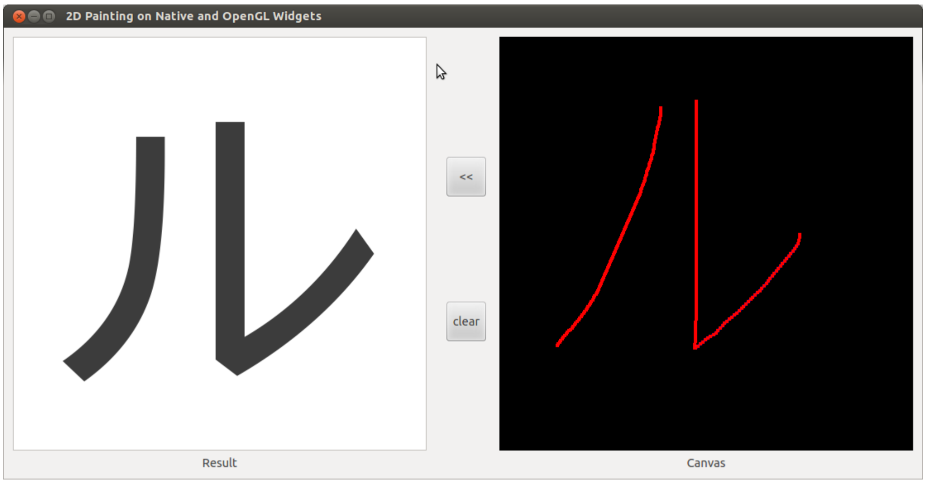


Figure 4.1: Character Recognition work overview

The idea for this project came from a previous work [Figure 4.1] with the idea of identifying handwritten japanese characters using stroke recognition methods.

We decomposed the Katakana characters into a set of strokes [Figure 4.2] that, when combined, could form any character. So, we wanted to recognize the user handwritten character by matching the strokes drawn to the templates we composed from the strokes in the set we constructed. To use the strokes matched to identify the character we introduced a restriction that the order of the strokes written had to follow the original stroke order of the character.

By decomposing the Katakana characters into a set of strokes, we built a tree structure that used the Japanese characters stroke order feature to enable the representation of the entire character set in a single tree, with each node being a stroke and each path from the root to a leaf making up a character.

The order restriction allowed us to represent the characters as a tree data structure [Figure 4.3]. In our tree, all nodes, except for the root and the leaves, were one of the strokes in our set. The tree we built had a void root in level 0, in level 1

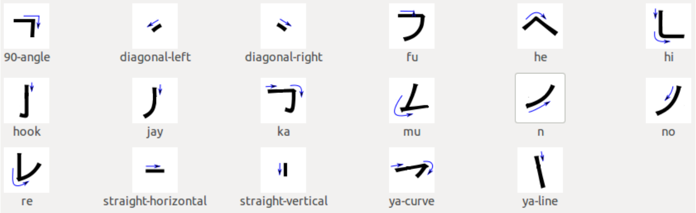


Figure 4.2: Katakana stroke set

were the nodes that represented the first stroke in a character order, in level 2 were the nodes that represented the second stroke in a character order, and so on for the other levels. The leaves are the characters that are represented by the nodes, in order, from the root to leaf. For the Katakana our tree had leaves in levels 2, 3, 4 and 5, as characters vary from 1 to 4 strokes.

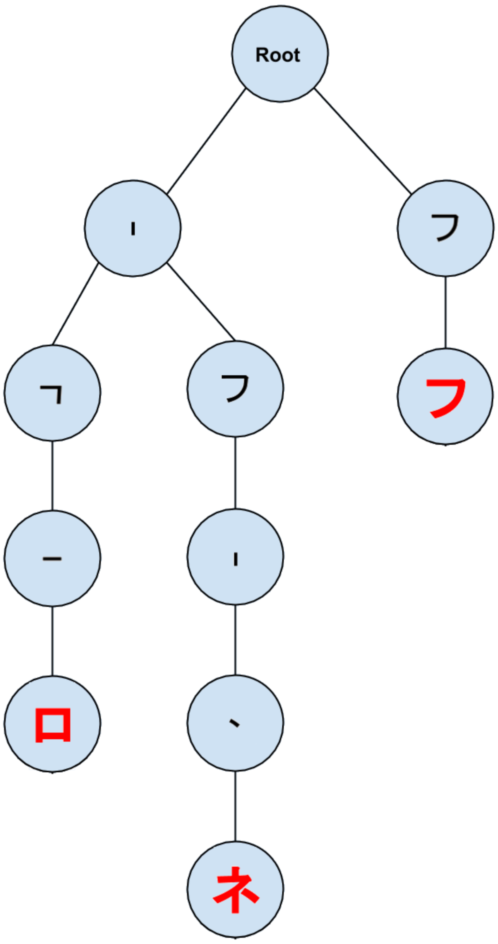
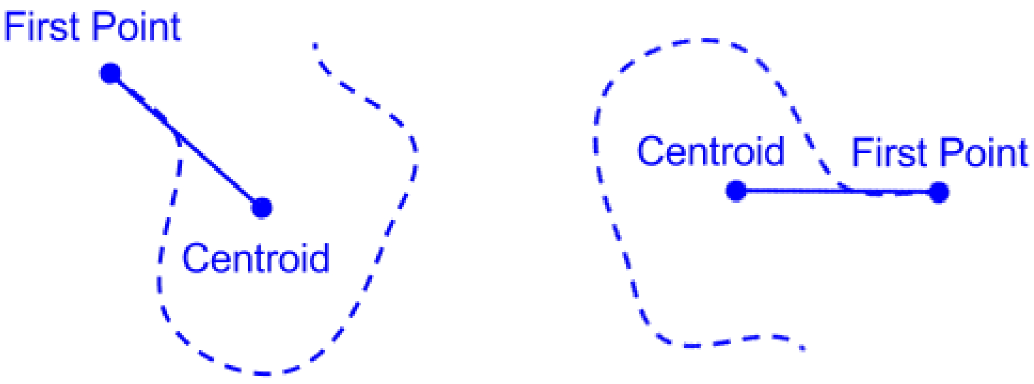


Figure 4.3: Example of representation of some characters in our tree

We first approached the solution by identifying each of the user written strokes, with the 1$ Recognizer [6] method and a template for each stroke that belonged to the Katakana set.

In the 1$ Recognizer a 4-step algorithm to match template strokes with input is



(a) Original drawing

(b) Drawing after rotation

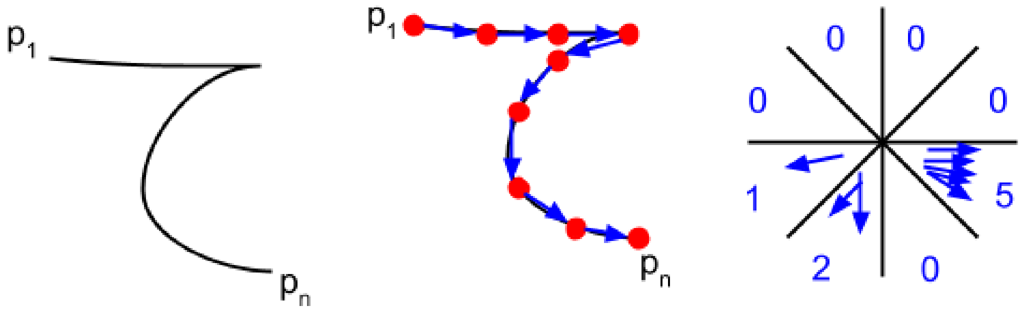
Figure 4.4: Example of the rotation step in the 1-dollar recognizer. On the left the original drawing and on the right the drawing after being rotated to have the angle between its first point and centroid at 0 °.

presented. First, either template or input, is resampled to a fixed number of equidistantly spaced points, so that it is possible to compare strokes drawn in different input movement speeds (sampling rates). The next step is alignment by performing a rotation. The stroke is rotated so that the angle between its centroid and the first point is at 0 degrees [Figure 4.4]. After, the stroke is non-uniformly scaled to a reference bounding box and then translated so that its centroid is at $(0, 0)$. Lastly, the candidate is compared to all stored templates, finding an average distance, that is converted to a [0..1] score. The highest score template is the matching result of the input stroke.

Our results varied for each character, but it was expected that some identifications would fail as we needed that different rotations of the same line represented different strokes. For instance, a horizontal straight line and a vertical straight line should be two different strokes, and the 1 dollar algorithm would recognize as the same template. To refine our algorithm we improved the solution implementing the approach described in the Angle Quantization method[7].

In the Fast Stroke Matching by Angle Quantization [Figure 4.5] matching of two strokes is done in the following way: Strokes are represented by directions, having every pair of neighbor points $p_i, p_{i+1}$ replaced with a vector $v_i = p_{i+1} - p_i$. Then, vectors are distributed in a unit circle divided in equal-size bins. A feature $Q$ is formed by adding up the number of vectors in each bin and normalizing so that each coefficient represents the percentage of stroke activity in that bin. Strokes are compared by evaluating the euclidean distance between their two features $Q$.

With the 1$ algorithm we had, given an input, a score for each template, and we were looking for the highest score template, indicating the best match. The problem



(a) The input stroke

(b) The stroke decomposed into vectors

(c) The vectors allocated in the bins

`(0,0,0,0,1,2,0,5)`

(d) Q feature representation

`(0,0,0,0,1/8,2/8,0,5/8)`

(e) Q feature normalized

Figure 4.5: Angle Quantization example

was that the difference in score between the highest, second and, sometimes, third highest was too tight to guarantee that it was the best possible match. So, we took the templates with the best scores from the 1 dollar algorithm and ran them through the angle-quantization technique, and used its score to aid in the decision of the best possible match.

With such, we've managed to correctly identify 67% of the Katakana characters, with a correct ratio of 89%, but we had very poor results for the other 33%. Some of our results are shown in Figure 4.6. The main reason was that even using both methods to find a solution there were still untreated ambiguities for the set of templates we had. Also, there were 2 characters in the Katakana that resulted in the same representation on our database tree.

After further research, we couldn't find another stroke recognition method to improve our algorithm and provide a solution with a high enough correct answer ratio. Through this conclusion, and the fact that character analyzer are seldom explored, we decided to change the focus of our work to the one we present in the following chapters.

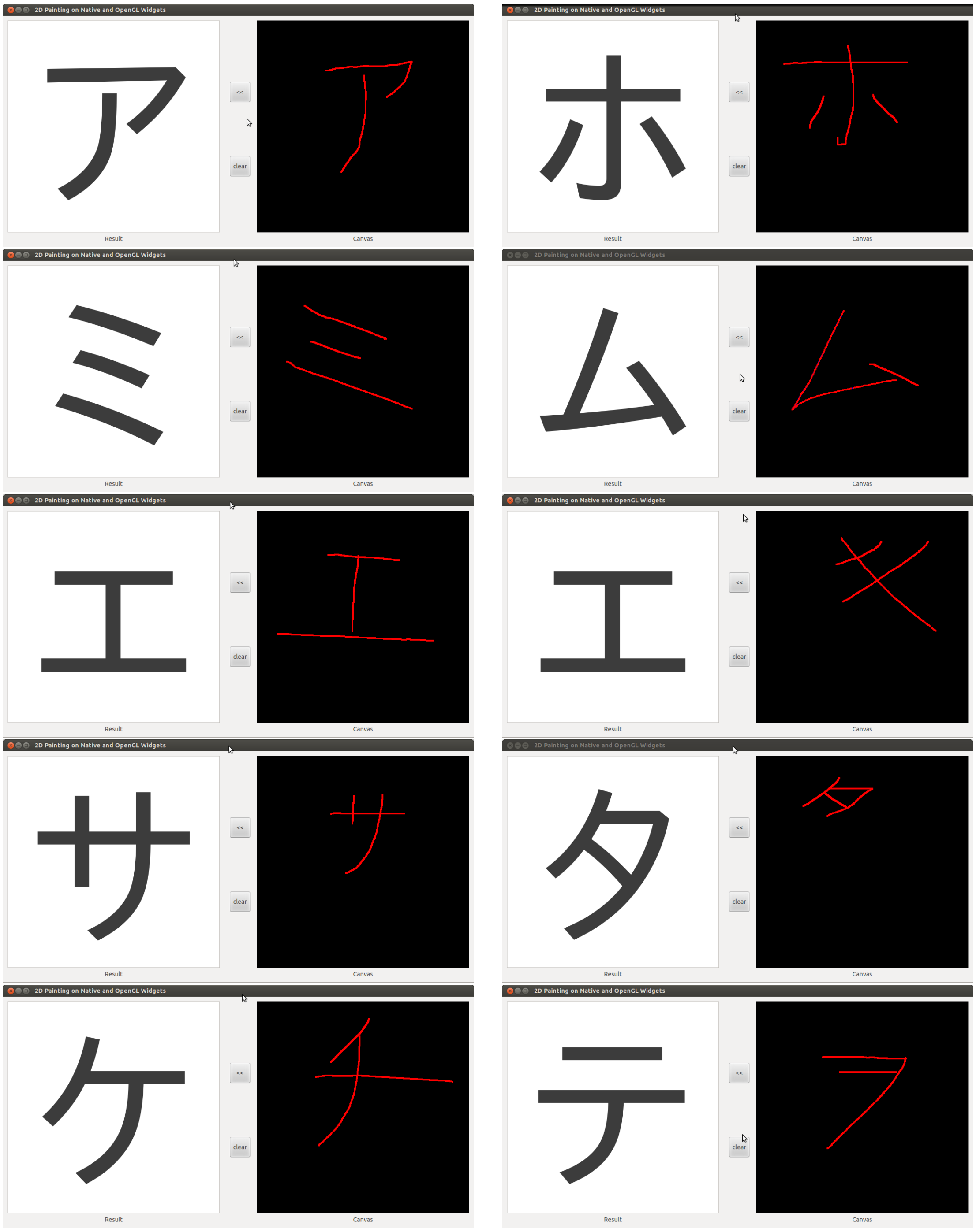
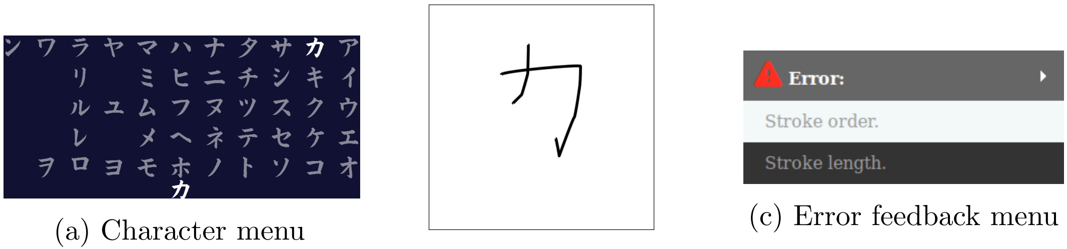


Figure 4.6: Character Recognition work examples. The two examples in the bottom row are incorrect recognitions made by the system.

# Chapter 5

# Input processing


ン ワ ラ ヤ マ ハ ナ タ サ カ ア
リ ミ ヒ ニ チ シ キ イ
ル ユ ム フ ヌ ツ ス ク ウ
レ メ ヘ ネ テ セ ケ エ
ヲ ロ ヨ モ ホ ノ ト ソ コ オ
カ
Error:
Stroke order.
Stroke length.

(a) Character menu

(b) Canvas with a drawing

(c) Error feedback menu

Figure 5.1: Technique overview

The application seeks to provide the user with the best possible feedback, one that can be spot on, easy to understand and that can lead the user to the corrections he needs to make. We compare a template of the character along side the drawing to extract and analyze the key features present in their structures to provide feedback to the user. Figure 5.1 represents an overview of the key interaction aspects of our system.

## 5.1 Drawing

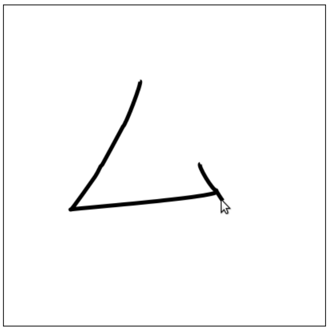


Figure 5.2: Drawing

The input is generated by drawing any number of continuous strokes [Figure 5.2]. The input is stored in two different data structures: one that holds the raw input points and one that holds cubic-Bézier points obtained through the Efficient Curve Fitting algorithm [10].

The Efficient Curve Fitting algorithm [10] produces a stroke represented as segments, where each segment is a cubic-Bézier curve. Its purpose in our application is aesthetic in order to produce a more attractive real-time feedback to the user by hiding possible noise and creating a smooth curve.

Before entering the testing stage, every raw data stroke is resampled as a polyline with a fixed number of points to match the resolution of template strokes.

## 5.2 Transformation

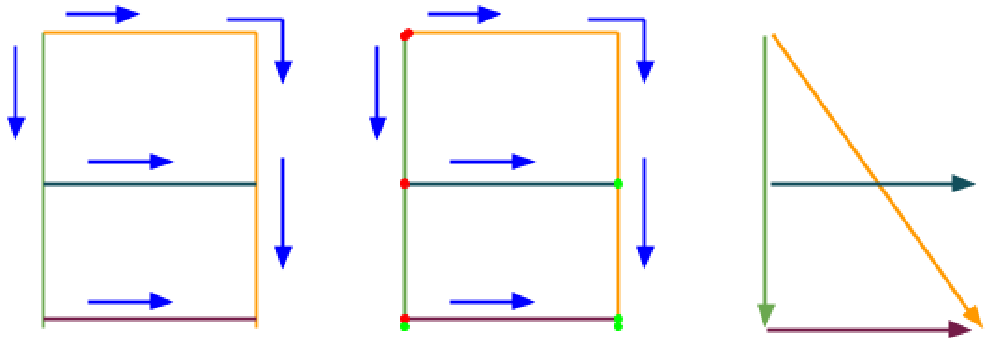


Figure 5.3: (a): The character with each stroke colored differently; (b): The same character with the begin point and end point of each stroke colored in red and green, respectively; (c): Vectors that define the orientation of each stroke.

The templates are composed of a set of strokes represented by polylines. We transform the input stroke to match the template, by applying a translation, rotation and isotropic scale. The reason for the isotropic scale comes from the fact that we do not want our transformation to correct any distortions the user made on his drawing, that would mislead our error analysis.

Since we resampled every input and template stroke to be a polyline with the same number of equally spaced points we match every point in the template to its corresponding point in the input. We match, following stroke order on the character and point order on the stroke: the first point in the first stroke of the template with the first point in the first stroke of the input, the second point in the first stroke of the template with the second point in the first stroke of the input, and so on for every point in every stroke. With this matching, we build a linear system.

However, before we build the matrices, we first need to make sure the stroke order is correct in the input and that every stroke orientation (point order) is also correct. To start, if the input has a different number of strokes when compared to its matching template the application identifies this as an error that is immediately reported to the user and no further analysis of this input is made, as it makes our matching impossible. If the number of strokes is the same in both input and template, the method continues by checking the orientation of each stroke. Since orientation is a feature that plays a key role in Japanese characters, incorrections are extracted, to be later presented to the user, and corrected in the input data, by inverting stroke orientation, to allow further analyses. Orientation of a stroke [Figure 5.3] is assumed to be the orientation of the vector that goes from the first to the last point of the polyline representing the stroke.

We build a column vector $b$ with every dimension of every point in the template in order from the first point in the first stroke to the last point in the last stroke, and a matrix $A$ with every point in the input, following the order in which they match the values in $b$. With such, we find the least-square solution with the singular value decomposition algorithm which gives us the transformation vector $x$ that solves: $Ax = b$.

To find a matrix that performs a rigid transformation as well as an isotropic scale, the system has to be built in a proper way. In order to obtain the isotropic scale we first normalize all the points in the input and also the points in the template with the matrix:

$$N = \begin{bmatrix} s & 0 & -\frac{c_x}{s} \\ 0 & s & -\frac{c_y}{s} \\ 0 & 0 & 1 \end{bmatrix}$$

where $(c_x, c_y)$ is the centroid point and $s$ is a scale so that the mean distance from the origin $(0, 0)$ is $\sqrt{2}$. Then, we build $A$ and $b$ so that we find:

$$x = \begin{bmatrix} \alpha \cos\theta \\ \alpha \sin\theta \\ t_x \\ t_y \end{bmatrix}$$

where $\alpha$ is the scale, $\theta$ is the rotation angle and $(t_x, t_y)$ is the translation vector. Since we have a column vector $b$:

$$b = \begin{bmatrix} x^t_{11} \\ y^t_{11} \\ x^t_{12} \\ y^t_{12} \\ \ldots \\ x^t_{nm} \\ y^t_{nm} \end{bmatrix}$$

where $(x^t_{nm}, y^t_{nm})$ is the point $m$ in stroke $n$ of the template. We build $A$ to form the correct system $Ax = b$:

$$A = \begin{bmatrix} x^i_{11} & y^i_{11} & 1 & 0 \\ x^i_{11} & y^i_{11} & 0 & 1 \\ x^i_{12} & y^i_{12} & 1 & 0 \\ x^i_{12} & y^i_{12} & 0 & 1 \\ & \ldots & & \\ x^i_{nm} & y^i_{nm} & 1 & 0 \\ x^i_{nm} & y^i_{nm} & 0 & 1 \end{bmatrix}$$

where $(x^i_{nm}, y^i_{nm})$ is point $m$ in stroke $n$ of the input.

From the least-square solution for the linear matrix system built with our matching we find vector $x$ and build the transformation matrix:

$$M = \begin{bmatrix} \alpha\cos\theta & -\alpha\sin\theta & t_x \\ \alpha\sin\theta & \alpha\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Then, we remove the normalization we did before:

$$M' = N_t^{-1} M N_i$$

where $N_t^{-1}$ is the inverse of the normalizing matrix for the template points and $N_i$ is the normalizing matrix for the input points. With $M'$ we can transform the points in input space to template space by applying it to every input point.

# Chapter 6

# Feature analysis

In this section we discuss how the application does the different types of key feature comparisons.

After transforming the input points into template space we are ready to compare them. Through a battery of tests we compare key features present in the topology and context of the Japanese alphabet. These features are: coherence, rotation, stroke length and stroke intersection properties. In addition, we define a heuristic to rank the strokes and form a hierarchy to assist our decision making on error analysis.

## 6.1 Character coherence

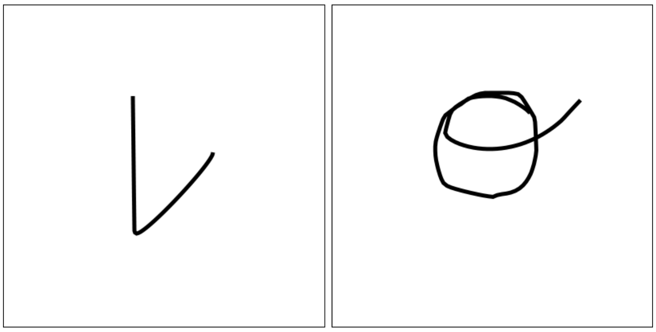


Figure 6.1: Coherence error example

An input-template pair with the same amount of strokes and all their paired strokes with the same orientation proprieties can still represent two different characters. Without proper handling, characters with more than one stroke might raise errors on the stroke length or intersection properties (although it is not guaranteed as some characters may be very similar), but single-stroke ones would slip by the analyses. This would happen because single-stroke characters will not raise length errors, as the stroke will always compose 100% of the character, and very few could raise intersection errors, as there is only a small group of characters with strokes that self-intersect.

To prevent the application from admitting any single stroke representation as a correct one for a single-stroke character, as well as capturing poor or completely incorrect drawings of multi-stroke ones, we created what we call a character coherence test [Figure 6.1].

In order to define character coherence we evaluate the "distance" from the template to the input. First, we define the distance between two strokes as:

$$S = \sum_{i=1}^{n} d_{p_i q_i}$$

where $d_{pq}$ is the Euclidean distance from point $p$ to point $q$, and $p_i$ and $q_i$ are the corresponding points, where the $X^{th}$ point of the $N^{th}$ stroke in a character is correspondent to the $X^{th}$ point of the $N^{th}$ stroke in the other character.

Character distance can them be defined as:

$$C = \sum_{i=1}^{k} S_i$$

where $S_i$ is the stroke distance calculated for the $k$ strokes of a character.

We compare the template-input distance to a threshold, if it exceeds it, we are faced with an input that is too far from the template to be considered as the same character.

When the coherence error is raised it is presented as the only error, since it makes further analysis impractical.

## 6.2 Rotation

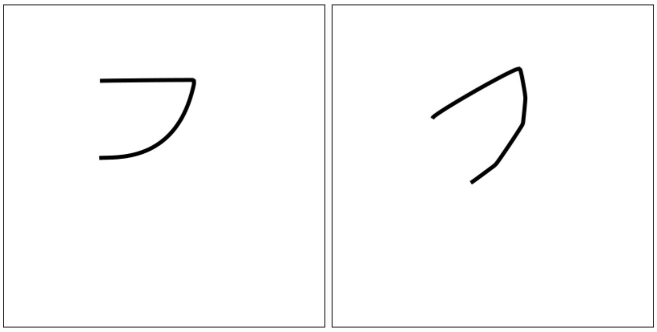


Figure 6.2: Rotation error example

Any rotation on the input can be fixed with the transformation to the template character space, but a representation too crooked or even upside down is interpreted as incorrect and so it needs evaluation [Figure 6.2].

The rotation from the template to the input can be extracted from the least-square solution vector found during the transformation step. From the $\alpha \cos \theta$ and $\alpha \sin \theta$ elements of the vector $x$ (presented in section 5.2) we can extract the angle $\theta$ of the rotation. The angle is compared with a threshold to possibly raise a rotation error. An animation is shown to explicit the nature of the mistake.

## 6.3 Stroke length

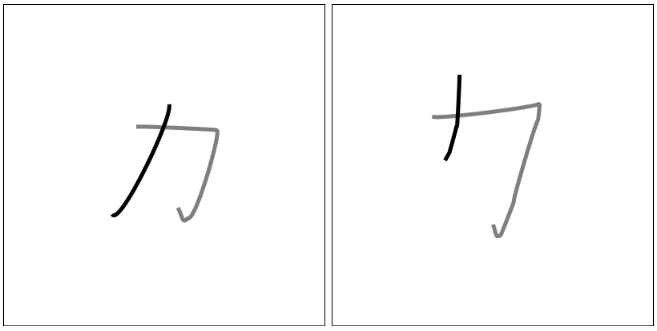


Figure 6.3: Stroke length error example

Proportion among strokes from the same character represent another important feature [Figure 6.3]. An out of proportion stroke length impacts the aesthetics and properties of a character and is usually indication of poor writing.

To evaluate the stroke length we define what we call character total stroke length. $Ls_i$ being the length of stroke $i$, total stroke length is then defined as:

$$TSL = \sum_{k=1}^{n} Ls_k$$

for a character with $n$ strokes.

We then proceed to calculate, for each stroke, how much does it contribute to the character total stroke length:

$$Cs = \frac{Ls_i}{TSL}$$

We compare $Cs$ for each stroke from the template with its counterpart from the input and allow the values to differ by a threshold. If the difference is higher than the threshold a length error is raised.

## 6.4 Stroke intersection properties

One of the most important features in Japanese character topology are the intersections. The properties of intersections among strokes are sometimes vital to differentiate two distinct characters[Figure  6.4].

(a) Character su

(b) Character nu

Figure 6.4: Two very similar katakana characters

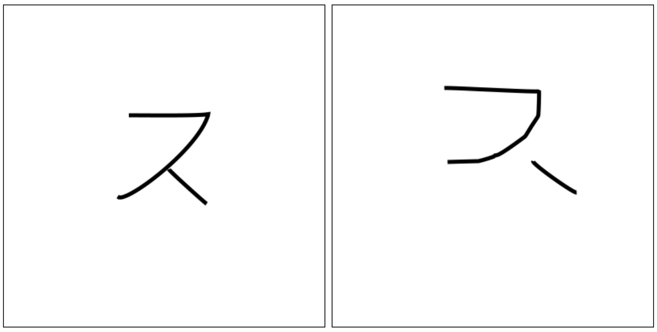


Figure 6.5: Non-existing intersection error example

Usually, two types of errors concerning intersections are made when drawing: Drawing character without an intersection it should have [Figure 6.5] and drawing an intersection that should not exist [Figure 6.6]. But we use the intersection properties to capture errors in the relative positioning among strokes as well. A misplacement of a stroke inside a character is easy to fix, but it is an error that has major relevance [Figure 6.7], as there are characters with very similar topology and strokes. So we evaluate where intersections occur on each stroke.

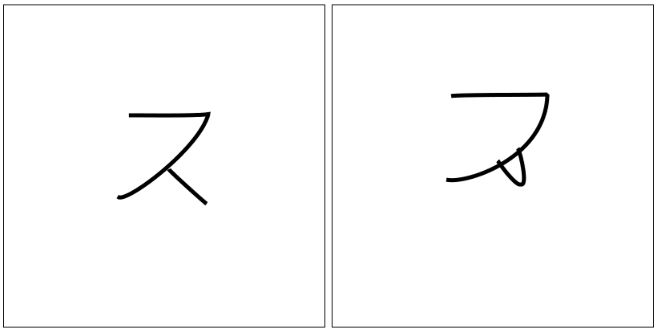


Figure 6.6: Too many intersections error example

The evaluation of intersections can then raise three types of errors: intersections that exist but are misplaced; intersections that are not drawn; and strokes that intersect one another more times than they should. For the last two, the opposite

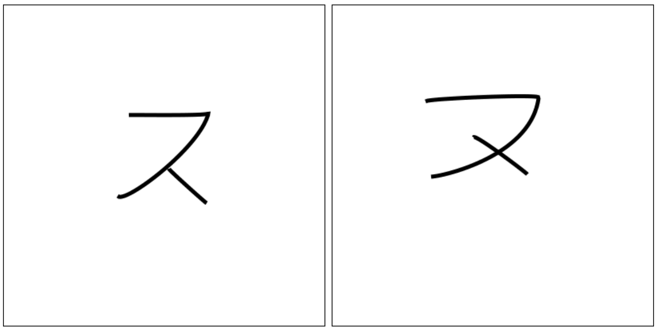


Figure 6.7: Misplaced intersection error example

is also considered. To manage the necessary tests, when extracting intersection data, we pinpoint the intersection location and, for every stroke involved, a value $0 \leq t \leq 1$, is obtained. $t = 0$ represents the first point of the polyline and $t = 1$ represents the last point of the polyline.

By checking for the existence of intersections on both the template and the input, extra intersections the input may have, and the difference on all pair values of $t$ against a threshold we are able to raise all possible errors concerning the intersection properties of a character.

When evaluating the presence of an intersection, we understand that it is possible to draw strokes very close to one another, meaning for them to intersect, but not actually connecting both [Figure 6.8], which is a nuisance for the user as distances of a few pixels are usually hard to spot on drawings. Thus, our algorithm checks for pairs of strokes that have points with an euclidean distance lower than a certain threshold, and maps them as existing intersections.

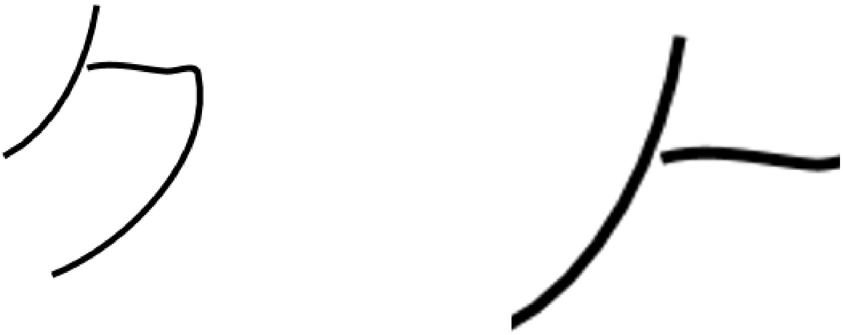


Figure 6.8: Two strokes that come very close, but never actually intersect.

## 6.5 Hierarchy

Multiple length errors are sometimes not an accurate feedback. Error in length may present itself as multiple entries in error log as characters have no size restrictions.

Sometimes it's not possible to say if stroke $x$ is too small or stroke $y$ is too big, in which case our system would raise length errors for both stroke $x$ and $y$. This same concept can be applied to stroke intersections. We designed a solution to choose the errors we present in these situations, not only to avoid this ambiguity but also to not overwhelm the user by showing too many errors.

As our method is proposed as a learning tool, it is necessary to evaluate what is the best possible feedback we can give the user. Multiple error entries related to the same error can generated confusion and redundancy, as they are usually only different points of view of the same error. The avoidance of aggressive error logs and related misconceptions is to refrain the feeling of a very negative feedback from the user, as we try to conduct the learning process as a gradual and satisfying experience.

By developing a concept of hierarchy among the strokes of a character, we try to find which stroke that, when fixed, has the most positive return on the character, i.e., the one that decreases the distance between the input and template the most. A multi-stroke character need to be evaluated as a whole, not only the strokes that represent it, but also their relationship (their intersections and length proportion). But, if we evaluate its individual strokes separately [Figure 6.9], a representation can be compromised by a poor drawing of a single stroke. We created a heuristic based on the length and intersection features. We calculate a score for each stroke and then rank them accordingly.

Suppose that $T^i$ and $I^i$ are characters composed only of stroke $i$ of the template and input, respectively. $I^i$ is then transformed into $T^i$ space. With $C(i)$ being the character distance (defined in section 6.1) between $T^i$ and the transformed $I^i$. We find:

$$x_1 = \arg\min_i [C(i)]$$

where $x_1$ is the stroke that alone has the best match with its correspondent.

We then create $T^{x_1,i}$ and $I^{x_1,i}$ which are characters composed only of strokes $x_1$ and $i$ of the template and input, respectively. Then $I^{x_1,i}$ is transformed into $T^{x_1,i}$ space. With $C(i)$ being the character distance between $T^{x_1,i}$ and the transformed $I^{x_1,i}$, we find which stroke fulfills the equation:

$$x_2 = \arg\min_i [C(i)], i \neq x_1$$

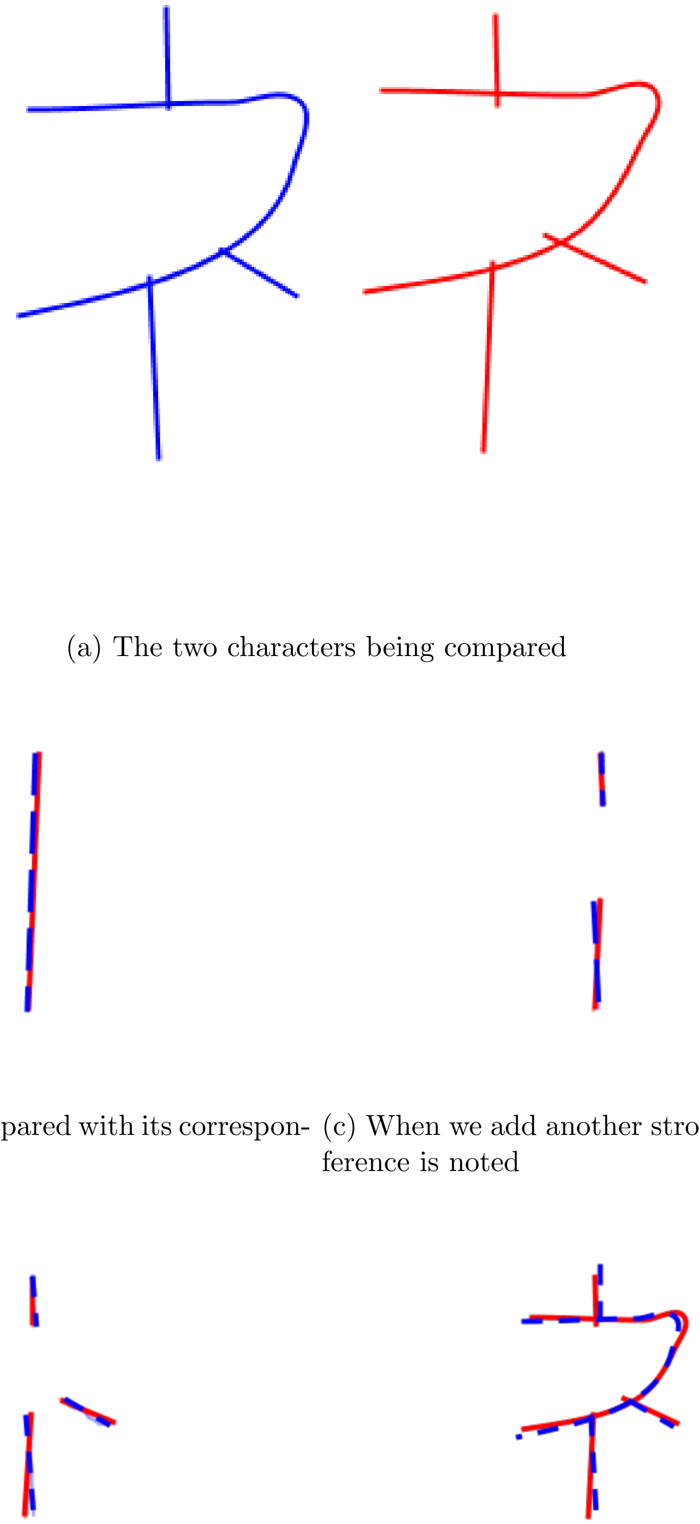
where $x_2$ is the stroke that when together with $x_1$ has the best match with its corresponding pair.

We continue the process by adding the stroke that belongs to the minimum character distance and repeating the above step until we build the entire character again. We then will have the hierarchy: $x_1, x_2, ..., x_n$. This process is represented in Figure 6.10.

Using this hierarchy, when multiple errors in length or intersection are raised, the user will only be presented with the ones concerning the highest ranked stroke [Figure 6.11].

## 6.6 Examples

Following are some examples of the results of our method [Figures 6.12 , 6.13].



(a) The two characters being compared

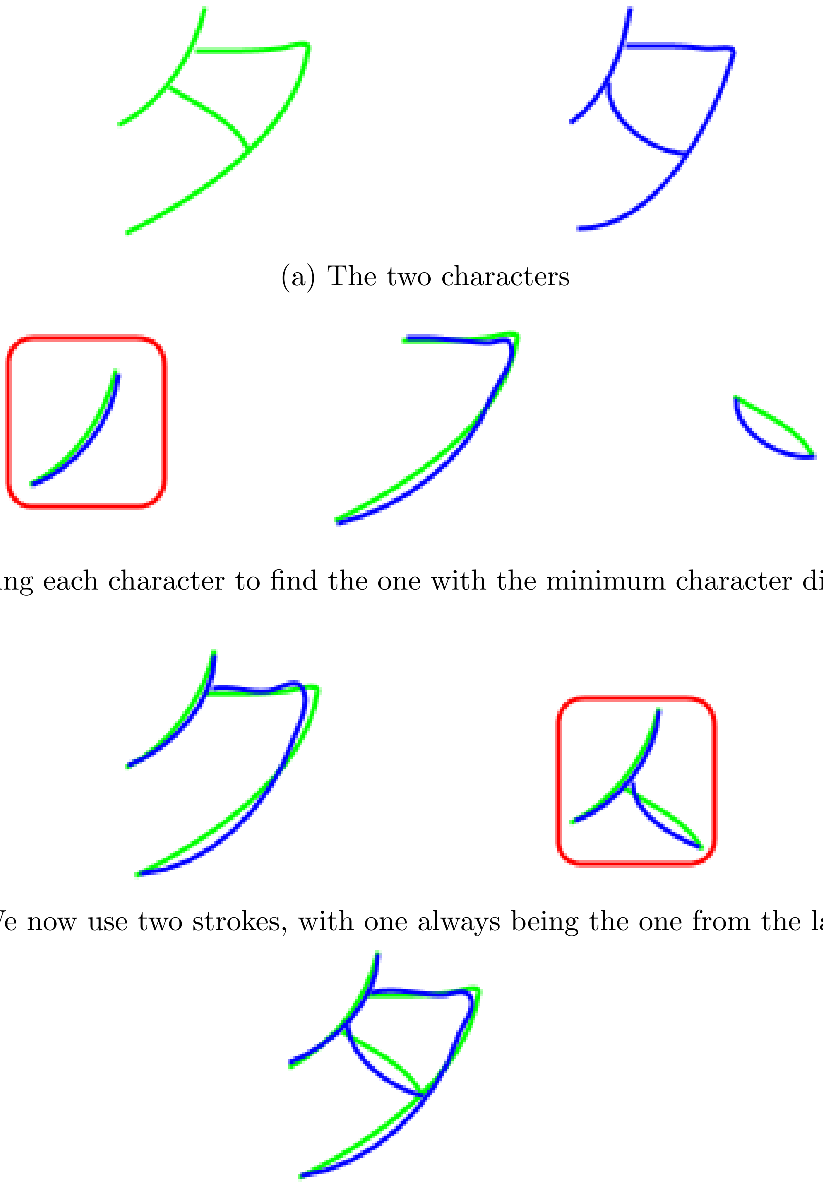
(b) The stroke, compared with its correspondent is very similar

(c) When we add another stroke, a little difference is noted

(d) A third stroke is added, but both characters are still very similar

(e) The full character has more notable differences

Figure 6.9: Hierarchy example



(a) The two characters

(b) Analyzing each character to find the one with the minimum character distance(in the red square)

(c) We now use two strokes, with one always being the one from the last step

(d) A third stroke is added, in this case the character is complete so the hierarchy has been calculated.

Figure 6.10: Hierarchy calculation example

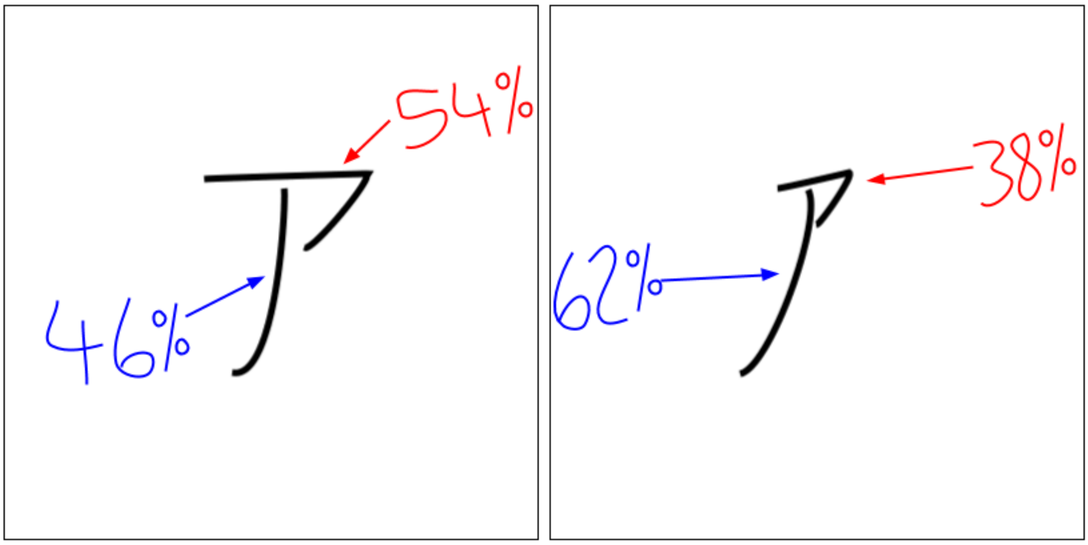
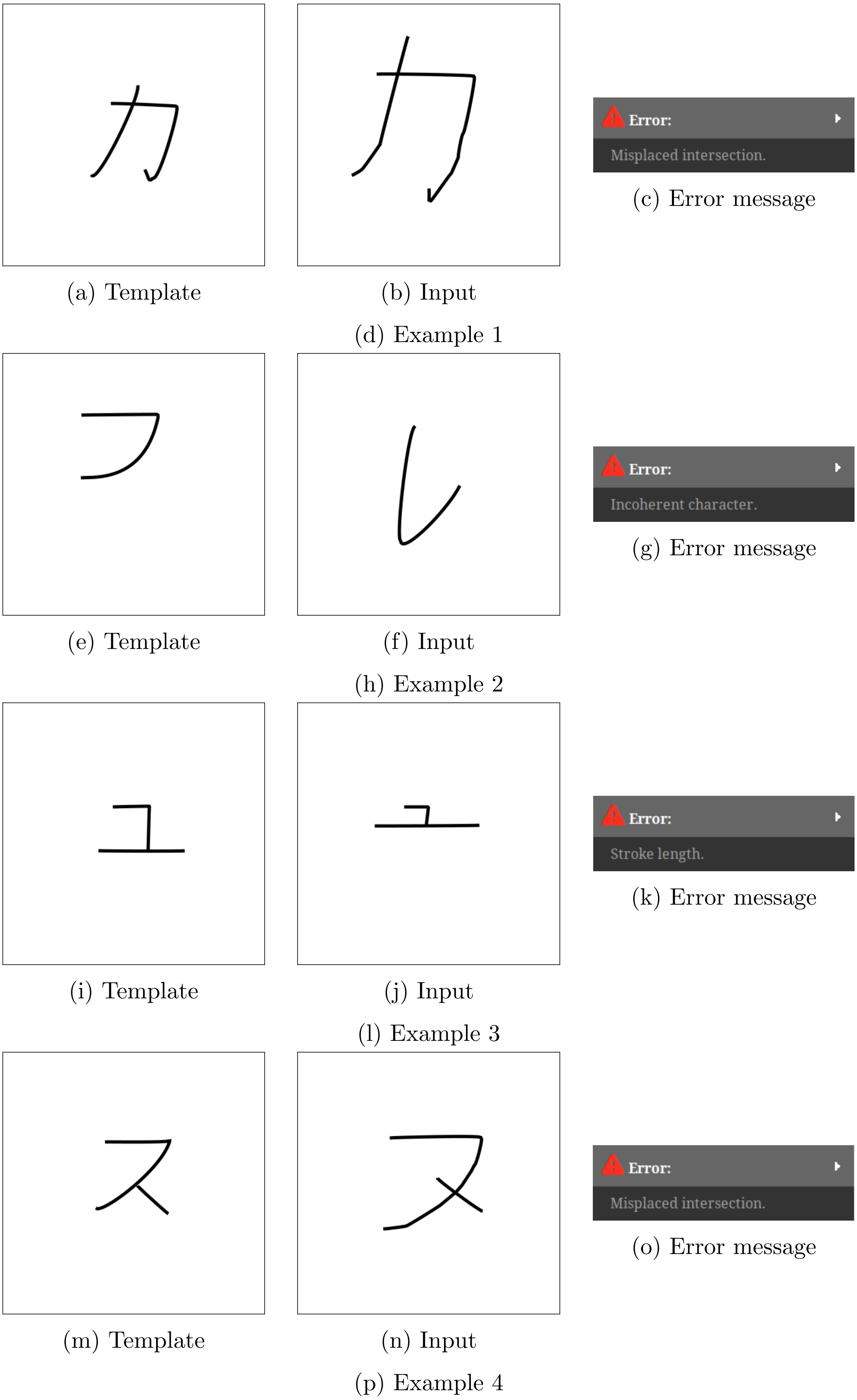


Figure 6.11: In this example both lengths, represented by the percentages, are out of the threshold and 2 errors would be raised, but using the hierarchy we will show only one. Clearly, fixing one would fix the other in this case.



Error:

Misplaced intersection.

(a) Template

(b) Input

(c) Error message

(d) Example 1

Error:

Incoherent character.

(e) Template

(f) Input

(g) Error message

(h) Example 2

Error:

Stroke length.

(i) Template

(j) Input

(k) Error message

(l) Example 3

Error:

Misplaced intersection.

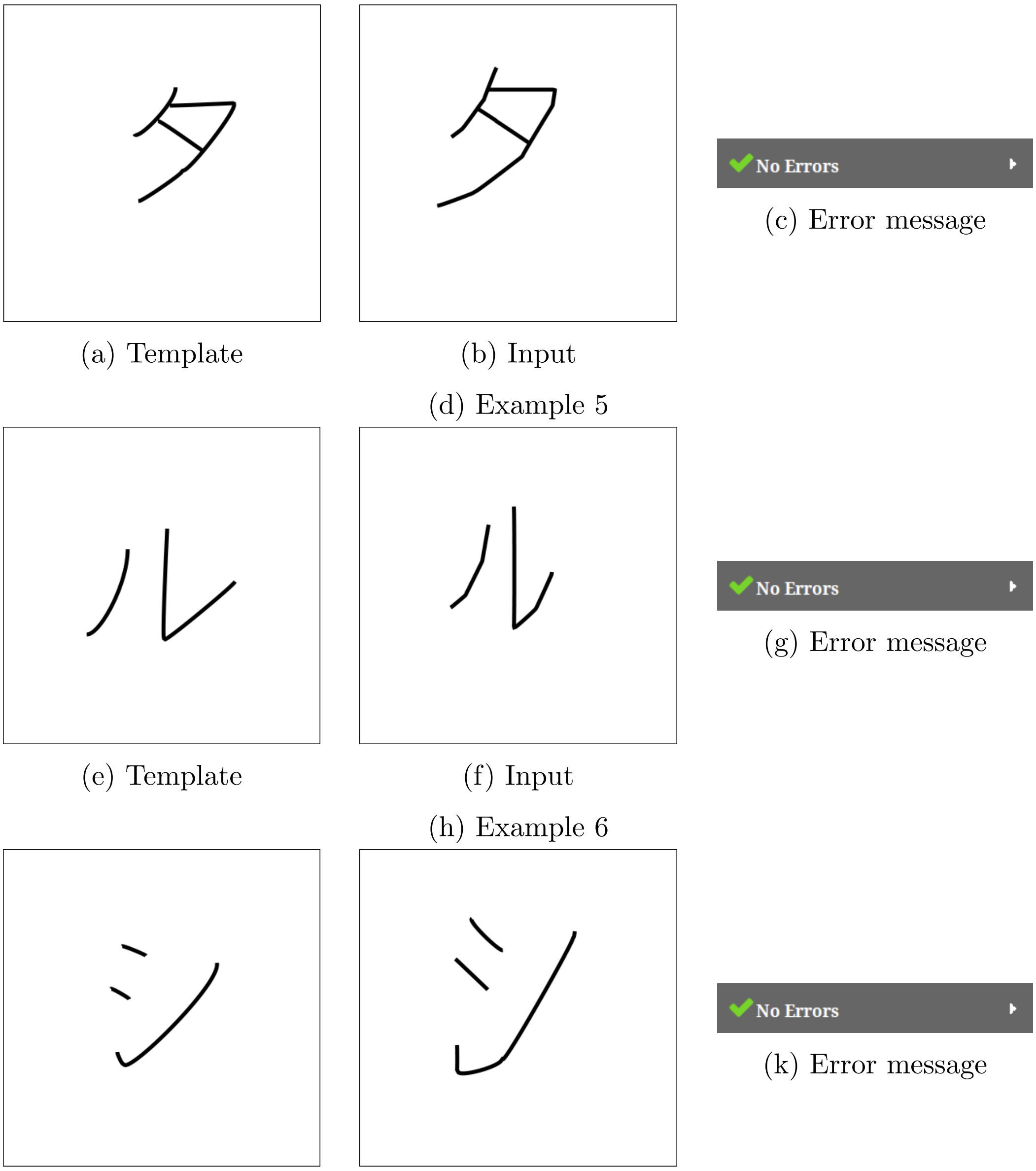
(m) Template

(n) Input

(o) Error message

(p) Example 4

Figure 6.12: Result demonstration



(a) Template

(b) Input

No Errors

(c) Error message

(d) Example 5

(e) Template

(f) Input

No Errors

(g) Error message

(h) Example 6

(i) Template

(j) Input

No Errors

(k) Error message

(l) Example 7

Figure 6.13: Result demonstration 2

# Chapter 7

# Implementation

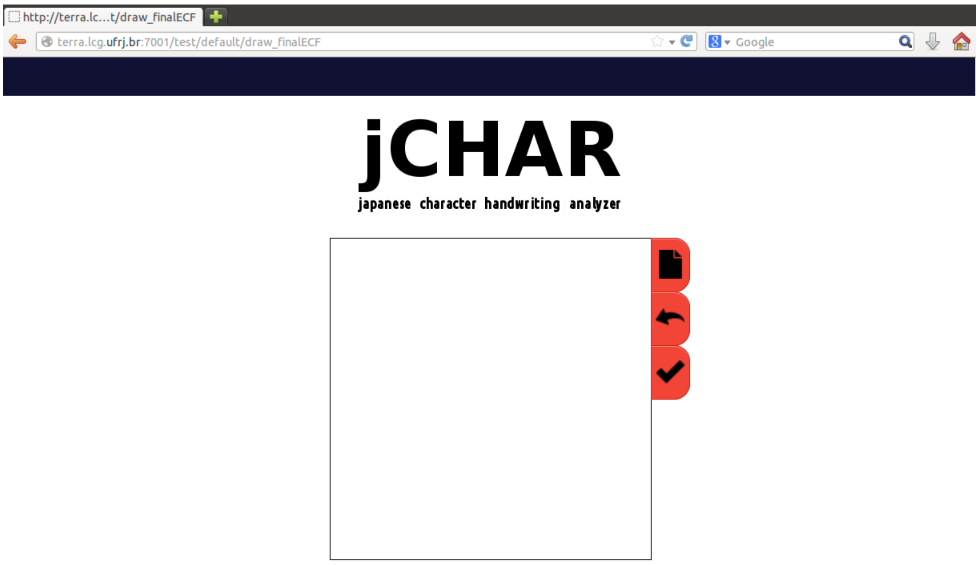

http://terra.lc...t/draw_finalECF
terra.lcg.ufrj.br:7001/test/default/draw_finalECF
Google
jCHAR
japanese character handwriting analyzer

Figure 7.1: Application interface

Our tool [Figure 7.1] is available as a web application implemented with html5 and Javascript along with a Python back-end server.

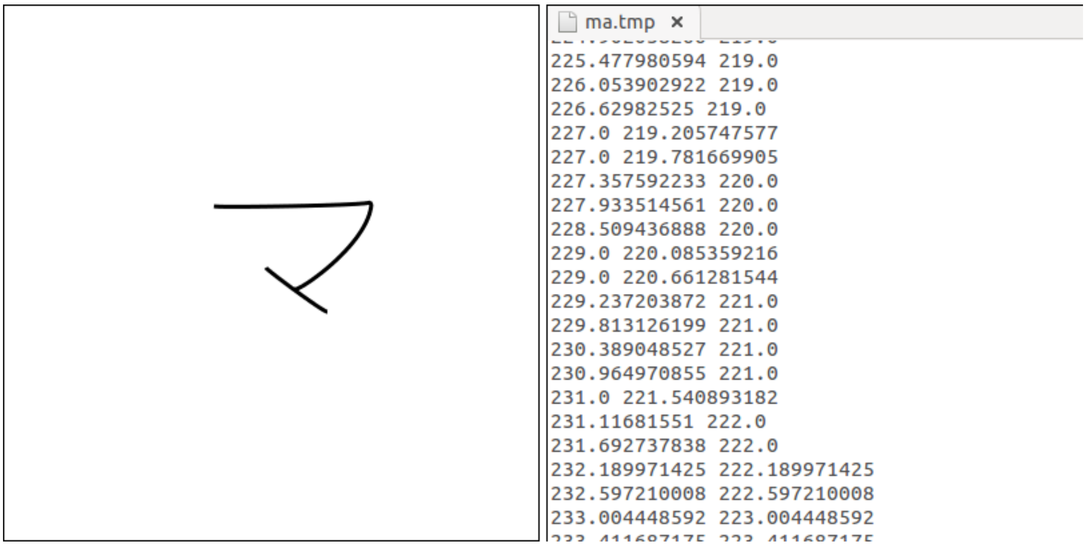
On the front page a html5 2D canvas implemented with the aid of Javascript is presented to the user. After choosing a character from the dropdown menu, he is able to freely draw with the mouse, a touchpad or trackpad. To assist him, buttons that allow undoing the last stroke and clearing the canvas are present. All input is treated with the Efficient Curve Fitting algorithm (presented in section 5.1) in real-time to give the user the perception of a smooth drawing.

When done drawing, the user submits it, sending the data to the server. The server does the analyses and redirects the user to a page where interactive feedback is displayed.

Feedback is presented as an interactive menu, listing the errors raised or a message informing that no errors were found. Upon clicking on any of the error messages animations are shown to guide the user on what corrections need to be made on his drawing.

## 7.1 Template Creation

In the same fashion to the user drawing input, a interface to capture template drawings was made. A 2D canvas is used to draw and a name is chosen for the template. The templates were named according to the character it represents. After drawing, the input data is submitted to the server, that stores the information on a file. Each template has its own file, that contains all the points that make up the strokes, the points that make up its Efficient Curve Fitting representation (only used when the character is shown to the user), as well as length and intersection information, to avoid the need of calculating them at runtime. An example of a template and its file contents in shown on Figure 7.2.



```
225.477980594 219.0
226.053902922 219.0
226.62982525 219.0
227.0 219.205747577
227.0 219.781669905
227.357592233 220.0
227.933514561 220.0
228.509436888 220.0
229.0 220.085359216
229.0 220.661281544
229.237203872 221.0
229.813126199 221.0
230.389048527 221.0
230.964970855 221.0
231.0 221.540893182
231.11681551 222.0
231.692737838 222.0
232.189971425 222.189971425
232.597210008 222.597210008
233.004448592 223.004448592
233.411687175 223.411687175
```

Figure 7.2: On the left: The template for the character ma. On the right: The contents of the file of this template on the server.

# Chapter 8

# Results and Discussion

We made our platform available for users to experiment with and give feedback on their experience. All tests listed here were made with the thresholds shown on the Table A.1.

For our tests, we made a database containing all of the Katakana characters. To better analyze the results, we categorized our users into groups, according to their knowledge level of the Katakana character. We decided to evaluate the user feedback using the disposition of users in these groups as we believe users under different groups can evaluate our approach from various perspectives, as they would benefit in different ways from our solution [Table 8.1]. Users were also asked what kind of input device they used for the tests [Table 8.2].

We had 16 users test the application. They were given a test script and access to the application and were asked to evaluate their experience under these aspects: Interface, feedback (from the system) and utility.

| | |
|---|---|
| I have a lot of experience with the Japanese language. | 0% |
| I have full knowledge of the Katakana, but I'm still studying/have not completed the studies of the Japanese language. | 19% |
| I know the Katakana, but I am currently learning/have not fully learned it. | 13% |
| I have some contact with the Japanese language, but never studied it. | 13% |
| I have no knowledge of the Japanese language. | 56% |

Table 8.1: Knowledge of the Japanese language

## 8.1 Interface

We asked the users to qualify the interface [Table 8.3]. Since we propose a learning tool, we find necessary that it has an accessible, intuitive and easy to use interface.

| | |
|---|---|
| Mouse | 50% |
| Graphics tablet | 44% |
| Touch interface | 0% |
| Other | 6% |

Table 8.2: Input device used

| | |
|---|---|
| Very satisfying. | 37% |
| Good, minor changes could be make it better. | 44% |
| Average, missing some options. | 19% |
| Poor, a lot of changes have to be made. | 0% |
| Awful, It ruined my experience with the application. | 0% |

Table 8.3: Interface results

To evaluate such interface features users ranked them, and were also asked if they would make any changes.

## 8.2 Feedback

| | |
|---|---|
| Great. Spot on and easy to understand. | 44% |
| Good, but the animation and text could use a little improvement. | 50% |
| Average, found it hard to understand sometimes. | 6% |
| Poor. I struggled to understand what it meant. | 0% |
| Not helpful at all. | 0% |

Table 8.4: Feedback results

We asked the users to give their opinion on the feedback the system gives them on their errors [Tables 8.4 and 8.5]. They were asked if they thought the feedback is clear, helpful and, most importantly, if the changes they needed to make to fix their drawings were sufficient for them to go back and try again.

## 8.3 Utility

We asked the users if they believe they are able to learn and improve their handwriting through the use of this platform [Table 8.6].

| | Perfect. The animation made the errors obvious. | Good. Need a few changes but satisfying overall. | Bad. It should be redone. | I never saw this type of feedback. |
|---|---|---|---|---|
| Number of strokes | 75% | 13% | 0% | 13% |
| Stroke orientation | 75% | 13% | 0% | 13% |
| Stroke coherence | 19% | 6% | 0% | 75% |
| Stroke length | 19% | 63% | 19% | 0% |
| Non-existing stroke intersection | 19% | 6% | 13% | 63% |
| Stroke intersection count | 13% | 13% | 19% | 56% |
| Misplaced stroke intersection | 13% | 44% | 31% | 13% |

Table 8.5: Specific feedback results

| | |
|---|---|
| Yes, It's a great tool for those learning it. | 44% |
| Yes, but needs some improvements. | 56% |
| No, I would not recommend it. | 0% |

Table 8.6: Utility results

# Chapter 9

# Conclusion

We presented, in this work, a Japanese character handwriting analyzer application. Our application compares geometric and topology features from a template with the user input to generate feedback of his drawing.

We proposed a method to evaluate handwriting through stroke properties. Our application is able to identify the most common and also the most serious mistakes made in Japanese character representation. Our work is presented as an educational tool for learning the written language, and provides a simple and easy to use interface and punctual feedback over the mistakes made in the users drawing. Our work also reduces the input noise through resampling and provides the users with better aesthetics with the implementation of the Efficient Curve Fitting algorithm.

The user feedback confirms the potential of our application, but also reveals some necessary changes to make it more accessible and a more efficient tool. As the target users of our application are those having little knowledge, we need to have a more clear feedback in some features critics, such as length and intersection errors, that did not fully please the majority of our test subjects.

The freedom provided by sketch-based drawing interfaces imply on difficulties in the analyses. Restraining such freedom would hurt the user experience and quality of the application, as our approach is to analyze hand drawings, so the capability to handle translations, scaling, rotations and distortions, not to mention the relationships between different parts of the drawing, is an essential part of our method.

When a person draws the characters frequently it is common to create some slight variations that adapt the character representation more to his style of handwriting, and thus creating some minor differences in representation that are not wrong by any means. As the method is now, some variations in the representation of a character are not well received by the application, and relaxing some of the thresholds used might make it so. However, our method is designed for those learning the language, so restricting the representations to their most formal shape is not a negative point.

Through the feedback the users gave us, we confirmed the potential of our platform as an educational tool that, given some minor changes, most specially in feedback representation of some features, can be a very useful tool for people learning the written language.

# Chapter 10

# Future Work

We would like to extend our testing database to the Kanji characters to explore the behavior of our platform with the higher complexity nature of those characters. Running an educational application on a web service environment does contribute to value our work as it greatly increase its accessibility. That being said, the high complexity Kanji characters, that can go up 29+ strokes in the more commonly used ones, and 60+ in some rare ones, would increase the computations made. In addition, there is room for improvement in our model, specially in how we handle the transformation from input to template space, as we need $n!$ combinations, where $n$ is the number of strokes, to check all possibilities regarding the order of the strokes, and also in its scalability. Another issue, in the computational point of view, would be the cost to build the hierarchy of character with this many strokes, as our algorithm has a complexity of $O(n^2)$, where $n$ is the number of strokes in the character.

We also believe that the user experience with our software can be improved by loosing the restriction on the intersection and length threshold, and also improving the feedback for the length and misplaced intersection features. We believe that these changes will reflect on even more positive user feedback.

Taking inspiration on some of the related work previously mentioned, another possibility could be making a interface for users, say a teacher, to create templates of characters proposed as homework and on another interface the students could draw the character given by the teacher, maybe even submit it as an assignment. On this scenario, no changes would need to be made on our method, only on its interface.

# Bibliography

[1] FORBUS, K. D., USHER, J. M. "Sketching for knowledge capture: a progress report". In: *IUI'02*, pp. 71–77, 2002.

[2] FORBUS, K. D., LOCKWOOD, K., KLENK, M., et al. "Open-domain sketch understanding: The nuSketch approach". In: *AAAI'04*, pp. 58–63, 2004.

[3] FORBUS, K. D., USHER, J. M., LOVETT, A., et al. "CogSketch: Open-domain Sketch Understanding for Cognitive Science Research and for Education". In: *SBM'08*, pp. 159–166, 2008.

[4] FIELD, M., VALENTINE, S., LINSEY, J., et al. "Sketch Recognition Algorithms for Comparing Complex and Unpredictable Shapes". In: *IJCAI'11*, pp. 2436–2441, 2011.

[5] EITZ, M., HAYS, J., ALEXA, M. "How Do Humans Sketch Objects?" *ACM Trans. Graph. (Proc. SIGGRAPH)*, v. 31, n. 4, pp. 44:1–44:10, 2012.

[6] WOBBROCK, J. O., WILSON, A. D., LI, Y. "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes". In: *UIST'07*, pp. 159–168, 2007.

[7] OLSEN, L., SAMAVATI, F. F., SOUZA, M. C. "Fast stroke matching by angle quantization". In: *ImmersCom'07*, p. article 4, 2007.

[8] HEROLD, J., STAHOVICH, T. F. "The One Cent Recognizer: A Fast, Accurate, and Easy-to-Implement Handwritten Gesture Recognition Technique". In: *SBM'12*, pp. 39–46, 2012.

[9] OTA, I., YAMAMOTO, R., SAKO, S., et al. "Online Handwritten Kanji Recognition Based on Inter-stroke Grammar", *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, v. 2, pp. 1188–1192, 2007.

[10] FRISKEN, S. "Efficient Curve Fitting". In: *Journal of Graphics Tools*, pp. 35–54, 2008.

# Appendix A

# Thresholds

| Threshold type | Value |
|---|---|
| Polyline point quantity | 128 |
| Character distance | 3000 * (number of strokes) pixels |
| Rotation angle | $\pi/8$ radians |
| Stroke length | 5% |
| $t$ | 0.05 |
| Intersection proximity | 14 pixels |

Table A.1: Application thresholds